USAGES SPECIAUX & PROGRAMMATION DU MACINTOSH EN PASCAL ET C

YANN FAIN

EYROLLES

USAGES SPECIAUX & PROGRAMMATION DU MACINTOSH EN PASCAL ET C

CHEZ LE MEME EDITEUR

G. LEVY - *L'outil MACINTOSH.*

M. DUCAMP - A. REVERCHON
- *Mathématiques et graphisme sur MACINTOSH.*

L. CARTY - C. VINCENT - E. ROY
- *Programmer en MACINTOSH.*

J.L. CHANTRY - H. DARMON - H. RABOURDIN
- *La pratique de CHART et MULTIPLAN sur MACINTOSH.*
50 graphiques et 6 applications de gestion.

K. JENSEN - N. WIRTH
- *PASCAL, manuel de l'utilisateur et rapport de définition.*

Ph. KRUCHTEN - *Le langage de programmation PASCAL.*

R.B. KIEBURTZ - *Introduction à la programmation avec PASCAL.*

L. LEON - *Programmation en Assembleur 68000.*

P. JAULENT - *Circuits périphériques de la famille 68000.*

F.R. MOORE - *Débuter en C.*

J.M. DRAPPIER - A. MAUFFREY
- *C par l'exemple.*

Ph. DAX - *Langage C.*

USAGES SPECIAUX & PROGRAMMATION DU MACINTOSH EN PASCAL ET C

YANN FAIN

EYROLLES

61, boulevard Saint-Germain – 75005 PARIS
1987

Si vous désirez être tenu au courant de nos publications, il vous suffit d'adresser votre carte de visite au :

Service « Presse », Editions EYROLLES,
61, Boulevard Saint-Germain
75240 PARIS CEDEX 05,

en précisant les domaines qui vous intéressent. Vous recevrez régulièrement un avis de parution des nouveautés en vente chez votre libraire habituel.

« La loi du 11 mars 1957 n'autorisant, aux termes des alinéas 2 et 3 de l'article 41, d'une part, que les « copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective » et, d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration, « toute représentation ou reproduction intégrale, ou partielle, faite sans le consentement de l'auteur ou de ses ayants droit ou ayants cause, est illicite » (alinéa 1er de l'article 40). »

« Cette représentation ou reproduction, par quelque procédé que ce soit, constituerait une contrefaçon sanctionnée par les articles 425 et suivants du Code pénal. ».

© Editions EYROLLES, 1987

TABLE DES MATIERES

TABLE DES MATIERES I

INTRODUCTION X

REMERCIEMENTS XIII

I. APERÇUS

COMMENT 1
DES SOURIS ET DES HOMMES 2
ATTENTION, ECOLE 3
TECHNOLOGIES 4
DE MEMOIRE DE MAC 6
SECRETS DE SORCIER 7
PROFESSION ? ASSEMBLEUR 7
32 QUESTIONS, 200000 REPONSES 8
TYPES DE DONNEES 9
SIGNEZ LE REGISTRE S.V.P. 9
QUELLE EST VOTRE ADRESSE? 11
FRANK LLOYD WRIGHT ET MACINTOSH™ 13
INDIVIDUS TYPES 16
DE LA ROUTINE 18
DES TRUCS DANS LE SYSTEME 20
Gérer les Gestionnaires dans la boîte à outils... 22
EN PARLANT AU FINDER 28

II. BLAISE ET VOUS...
LA VISITE GUIDEE DU PASCAL

PASCAL-IFORNIE 31
QUEL EST VOTRE NOM? 34
PROCEDURIERS 35
VOUS ETES FICHE... 37
DES POINTEURS, POINTES, ET LEURS "^ENFANTS" 40
TYPES DE TYPES 42
HALLELUJAH, HAENDEL! 44
ON DEMANDE: ORD4, POINTER, @ 46

MON COMPILATEUR? UN DETYPEUR! 50
MACPASCAL 51

III. CAMP DE BASE Nº 1: repos, nourriture, SHERPAS

REVUE 54
COMMENTS ET POURQUOIS:
DES GESTIONNAIRES
DE LA BOUCLE D'EVENEMENTS 54
DES RESSOURCES 56

IV. MES MEMOIRES

EMPILEZ-MOI CE HEAP, MERCI 57
HAPPEZ CETTE PILE, QUOI! 58
DEREFERENCER UN(E) HANDLE 63
STRUCTURES DE BLOCKS ET MASTER POINTERS 65
UN PEU PLUS DE GESTIONNAIRE DE MEMOIRE 66
RE-DETYPAGE 67
NEWPTR, DISPOSPTR 69
NEWHANDLE, DISPOSHANDLE 70
EMBOUTEILLAGE MEMOIRE 70
ERREURS COMMUNES, SUGGESTIONS 71
BOULOT DE SERRURIER: LOCK, UNLOCK 71
DES MACS 73
LE SEGMENT LOADER 73
UnloadSeg 74
GetAppParms 74
Jump Table 75
Chain 75

V. QUICKDRAW, SHOWBIZ ?
NO SIR, DISPLAYBIZ! 79

VOTRE ATTENTION, S.V.P. 80
CONCEPÇION 80
GRILLE 80
POINT 81
RECTANGLE 82

REGION 83
BITIMAGE 84
BITMAP 84
baseAddr 85
bounds 85
PATTERN 87
CURSEUR, HOTSPOT 87
LES 17 TRAVAUX DE QUICKDRAW 90
CHOISIS TON CHEVAL: GRAFPORT... 91
device 93
portBits 93
portRec 93
visRgn 93
clipRgn 93
bkPat, fillPat 94
pen (size, mode, ...) 94
txfont 95
txface 95
OBSESSION SETSUELLE 95
txmode 97
txsize 97
spExtra 97
color 97
patStretch 97
picSave, rgnSave, polySave 97
grafprocs 97
exemples de programmes 98

VI. PLUS SUR QUICKDRAW 103

DEFINITIONS 104
LUMIERES SUR LE PORT 105
Plus de rowWidth 106
InitGraf 106
OpenPort, InitPort, ClosePort 107
SetOrigin, SetPort 108
modes de transfert 109
Copy, Or, Xor, Bic 109
Pictures, Polygones 110
StuffHex 110

Appels divers 113
exemples de programmes 114

VII. LE FINDER, LES RESSOURCES 117

OBJETS DU FINDER 118
SUR LE BUREAU 119
LE FINDER, C'EST LA VIE 119
Fenêtres, Icones, Folders: caractéristiques 119
RESSOURCES 121
TYPES DE RESSOURCES / RMAKER 123
RESOURCE MANAGER 132
ROUTINES DU RESOURCE MANAGER 134
VEUILLEZ SIGNER ICI "CREATEUR" / RMaker 135
VERSION DATA, FREF, BUNDLES 138
ANALYSE EN HEXADECIMAL
(exemples de programmes) 140

VIII. EVENEMENTS 145

PRIORITE D'EVENEMENTS, POSTING 149
STRUCTURE DES EVENEMENTS 151
Le champ "message" 152
Le champ "modifiers" 153
Masques 155
GetNextEvent 156
APPELS DIVERS 157
GetMouse, GetKeys, Button, WaitMouseUp
et StillDown 157
EventAvail 158
PostEvent 158
FlushEvents 158
TickCount 159
SetEventMask 159
Echantillon de boucle d'événements 160
Exemples de programmes ("C", Forth) 161

IX. FENETRES ET CONTROLES **165**

CONTENU DES FENETRES 165
TOUTES CES FENETRES 167
LE WINDOW RECORD 169
EVENEMENTS ET FENETRES 170
AFFICHAGE DE FENETRES/UPDATE EVENTS 171
BeginUpdate 171
EndUpdate 172
AU TOTAL 172
ROUTINES DU WINDOW MANAGER 174
InitWindows 174
NewWindow 174
GetNewWindow 174
FindWindow 174
DrawGrowIcon 175
TrackGoAway 175
CloseWindow 175
DisposeWindow 175
HideWindow 175
ShowWindow 175
FrontWindow 175
LES CONTROLES 176
CONTROL RECORD 179
ROUTINES DU CONTROL MANAGER 180
GetNewControl, Newcontrol 181
DisposeControl 181
KillControls 181
FindControl 181
TrackControl 182
DrawControls 182

X. DIALOGUES ET MENUS **183**

POURQUOI DES DIALOGUES? 183
Exemples de programmes 184
LE DIALOG MANAGER 189
GetNewDialog 190
NewDialog 190
ParamText 191

EVENEMENTS DANS LES DIALOGUES 192
ModalDialog 192
isDialogEvent 192
DialogSelect 192
DAVANTAGE DE DIALOG MANAGER 192
DlgCut 193
DlgCopy 193
DlgPaste 193
DlgDelete 193
Alert 193
StopAlert, NoteAlert, CautionAlert 193
CouldDialog 193
FreeDialog 193
MMMMENUS 194
LE MENU MANAGER 196
InitMenus 196
NewMenu 196
AppendMenu 196
GetMenus 197
InsertMenus 197
DrawMenuBar 197
MenuSelect 197
MONPROG
(Premier exemple, utilisant un compilateur
Pascal convenable) 198
DAVANTAGE SUR LE MENU MANAGER 209
ClearMenuBar 210
SetMenuBar 210
MenuKey 210
exemple de programme en "C". Menu, boucle 211
PASCAL TML: MISE EN PLACE 217

XI. TEXTEDIT,
GESTIONNAIRE DE "JEUX DE CARACTERES" 223
JUSTIFICATION 224
LIMITES DE TEXTEDIT 225
EDIT RECORD 226
ROUTINES DE TEXTEDIT 227
TEInit 227
TENew 227

TEIdle 227
TEClick 227
TEKey 228
TEActivate 228
TEDeactivate 228
TEUpdate 228
TECut 228
TECopy 228
TEPaste 228
TEDelete 228
TEInsert 228
TESetJust 229
TESetSelect 229
TESetText 229
TECalText 229
TEGetText 229
TEDispose 229
AINSI FONTS FONTS FONTS LES POLICES DE CARACTERES 230
A L'ECHELLE 232
FONT RECORD 233
QUICKDRAW ET LE TEXTE 233
ECHANGE DE DONNEES AVEC QUICKDRAW 234
ROUTINES DU FONT MANAGER 235
GetFontName 235
GetFNum 235
realFont 236
SetFontLock 237

XII. UN TRAITEMENT DE TEXTE, DEUX FENETRES 237
Programme (Pascal, deux fenêtres) 254
TRANSPORT DE REGIONS 265

XIII PRINT, SCRAP, E/S, SON **276**

PRINT MANAGER 277
OpenPPort 277
PrintDefault 278

PRValidate 278
PRStlDialog 278
PRJobDialog 278
PROpenDoc 278
PROpenPage 278
PRClosePage 278
PRPicFile 278
PRCloseDoc 278
PrClose 278
LE MYSTERIEUX PRESSE-PAPIERS 278
SCRAP MANAGER 279
InfoScrap 279
ZeroScrap 280
GetScrap 280
PutScrap 281
FILE MANAGER 281
FSOpen 281
SFGetFile 282
SFPutFile 282
FSRead 283
FSWrite 283
FSClose 283
FlushVol 283
Eject 283
GetVol 283
GetVInfo 283
SetVInfo 283
GetFInfo 283
SetFInfo 283
GetFPos 283
SetFPos 283
DEVICE MANAGER 284
OpenDriver 285
Control 285
Status 285
KillIO 285
COMMUNICATIONS 285
RAMDOpen 286
RAMDClose 286
SerReset 286
SerHShake 286

SerSetBrk 287
SerCirBrk 287
SON 288
SIGNAL CARRE 288
SWSYNTHREC 288
StartSound 289
FTSYNTHREC 289
FFSYNTHREC 289
SoundDone 386
StopSound 289
SetSoundVol 289
GetSoundVol 289

XIV. LE COIN DES DEVELOPPEURS ET UTILISATEURS 290
'FICELLES' 290
Exemples de programmes, fonctions et procédures 388
OU OBTENIR DES RENSEIGNEMENTS 293
DEBUGGING 294
AUTRES OUTILS: LE RESOURCE EDITOR, SETFILE 295

APPENDICE A (Adresses) 298

APPENDICE B (Types, mémoire, Seg Loader) 300

APPENDICE C (QuickDraw) 302

APPENDICE D (Evénements OS Utilities, Vert Retrace, OS Dispatch, utility core, queue) 307

APPENDICE E (E/S, Son, Erreurs Système) 309

APPENDICE F (Font Mgr, Window Mgr, TextEdit) 313

APPENDICE G (Resource Mgr, Control Mgr) 316

APPENDICE H (Dialog Mgr, Desk Mgr, Menu Mgr, scrap Mgr) 319

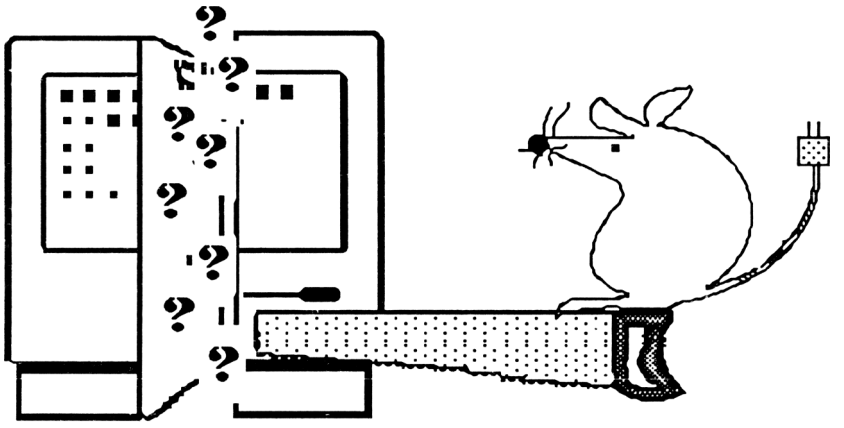
REMERCIEMENTS

J'aimerais d'abord remercier Monsieur Tissier, des Editions Eyrolles, pour sa contribution à l'élaboration de ce livre.

Ensuite, à un ami de toujours, Alain Drozd, Président d'International Computer.

Et, bien entendu, mon outil de travail quotidien, le Macintosh™, qui m'a aidé sans jamais se plaindre des horaires que je lui imposais.

INTRODUCTION



PLUS DE MYSTERE...

Ce **livre** est d'abord dédié à tous ceux qui veulent **comprendre** les secrets de leur **Macintosh™**. Après en avoir exploré l'intérieur, vous saurez comment gagner beaucoup de temps en évitant dans la plupart des cas le "Couper-Coller", en sachant utiliser l'utilitaire **SetFile**.

Vous pourrez ainsi transférer de l'information d'une **application** à une autre, rendre les fichiers créés sous **Overview** digestibles par d'autres applications. Ne perdez plus de temps en revenant au **Finder**.

Vous pourrez **transformer** les logiciels que vous achetez en sachant vous servir du **ResourceEditor**, ou de **ResEdit**. Et ainsi, **personnaliser** vos logiciels, **modifier** les textes affichés à l'écran, et les rendre adaptés à **vos** standards.

Mais ce n'est pas tout.

Apprendre à utiliser les logiciels en peu de temps, résoudre les mêmes tâches quotidiennes en moins de temps, c'est ce que nous attendons tous de la technologie.

Nous avons décidé de nous intéresser au **Macintosh**™ car précisément, il apporte ces deux éléments. Mais la facilité d'utilisation est accompagnée ici d'une programmation plus complexe.

Même si vous n'êtes pas un programmeur chevronné, vous avez en main un livre qui vous permet **aussi** d'écrire des applications, de comprendre les **Menus**, les **Fenêtres**, les **icônes**, les rapports entre celles-ci et les noms des applications, les **Ressources**, les **événements**, et de composer du **code** sur votre Macintosh™.

Les programmes fournis sont **OPERATIONNELS**. Ils sont assortis d'explications pratiquement à chaque ligne. Vous trouverez même un traitement de texte avec DEUX fenêtres différentes.

En tout premier lieu, vous verrez quels sont les éléments constitutifs du Macintosh™. Nous vous éclairerons sur les "pourquoi" de l'architecture de la machine. Nous rappellerons les principales règles du Pascal, et de "C", et quelques détails de l'assembleur du 68000.

Alors, vous saurez ce que signifient les mots les plus mystérieux, tels: Heap, Stack, Handles, Pointers, ORD4, ^^, Hlock, "déréférencer", et "détyper". Ensuite, vous entrerez immédiatement dans la partie logicielle de votre **Macintosh**™: les **gestionnaires**.

Un tel livre vous sera aussi utile si vous voulez comprendre l'intérieur d'autres machines, comme **l'Amiga** (sur lequel nous avons déjà développé deux logiciels aux Etats-Unis) ou **l'Atari** 1040 ST. Nous connaissons certaines des stratégies d'Apple, Atari, et Commodore, et leurs futurs respectifs, sinon communs, seront fort similaires.

Les cours fournissant un tel enseignement sont assez chers (de l'ordre de 4000 Fcs français), et ne couvrent pas la plupart des gestionnaires. La plupart s'arrêtent aux Menus et aux événements.

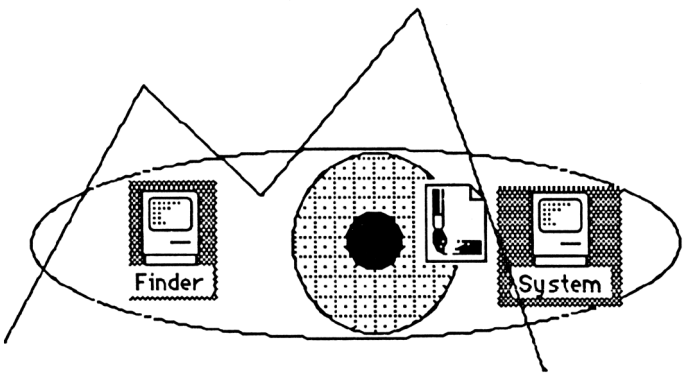
Si vous décidez de lire et relire "Inside Macintosh™", le "bottin" des développeurs, vous pourrez éventuellement attraper une bonne migraine, mais pas la science du Macintosh™.

Nous croyons avoir cette connaissance du Macintosh™, après bientôt deux années de développement, d'abord sur Lisa, ensuite, sur Mac.

Mais nous espérons surtout être bons pédagogues et, au cours de ce livre, vous faire connaître et assembler toutes les notions nécessaires à l'utilisation optimale de votre Macintosh™ et/ou sa programmation.

CHAPITRE I

APERÇUS



COMMENT...

Ce livre est composé de deux parties: les explications, et les programmes. Ceux-ci comprennent des commentaires.

La partie "explications" est constituée de sept chapitres. Dans chaque chapitre, cependant, vous trouverez des exemples de programmes illustrant le contenu de ce chapitre.

Le premier chapitre passe en revue le matériel et les composants logiciels du Macintosh™.

Le second chapitre couvre les aspects techniques de la programmation en Pascal (et en d'autres langages).

Au chapitre III, nous expliquons les "comment" et les "pourquoi" des options prises par les constructeurs du Macintosh™.

Que trouve-t-on dans le Chapitre IV? Ah, oui! Il est consacré à la gestion de la mémoire, et au Segment Loader (Chargeur de segments).

Les deux Chapitres suivants sont dédiés à QuickDraw, ses concepts, ses performances.

Puis, on s'attaque aux autres gestionnaires, aux "Ressources", aux "Evénements", tous sujets nécessaires pour aborder les programmes commentés.

Enfin, nous mentionnerons les environnements de développement existant à date sur le Macintosh™ (Avril 1986), et montrerons l'emploi d'utilitaires courants.

DES SOURIS ET DES HOMMES...

Il y a maintenant dix ans naissait dans un garage une petite société. Le succès fabuleux de la gamme Apple II prouva très tôt que les microordinateurs seraient bientôt omniprésents.

Après l'Apple II vint l'Apple III qui ne fut pas un succès.

En lançant son PC en août 1981, IBM donna à la microinformatique la crédibilité qui lui manquait pour être reconnue comme un outil de travail.

Ni l'Apple III, ni l'IBM PC n'amenaient de réelle révolution. Le Lisa®, par contre, avec ses logiciels intégrés, ses routines graphiques, la souris, le bureau, les menus, apporta un vent nouveau. On pouvait l'utiliser très vite, sans apprentissage, pratiquement, et ce, sans apprendre de langage.

Mais vous savez ce qui arriva à Lisa®: trop cher, cet ordinateur ne connut pas le succès mérité.

La plupart des caractéristiques du Lisa®, on les retrouvait dans les modèles Alto et Star, de Xerox, mais à un prix astronomique.

Il y a plusieurs moyens de positionner un curseur sur un écran: le stylo, la souris, les quatre touches (gauche, droite...), le joystick, et le doigt qui pointe sur l'écran. De tous ces moyens, la souris est la plus ergonomique (entendez celle qui requiert le moins d'effort).

Douglas Engelbart, père de la souris, voyait déjà en 1951 (il avait alors 26 ans) l'ère des microordinateurs. En 1964, travaillant pour le Stanford Research Institute (SRI), le mieme institut qui testa les facultés parapsychologiques d'Uri Geller, il mit au point la souris pour sélectionner des portions de texte sur un écran. La souris d'alors était faite de bois sculpté, et comportait deux roues perpendiculaires.

Les gens du SRI avaient même essayé de contrôler le mouvement du curseur grâce au...genou. La vie était douce au PARC (Palo Alto Research Center), mais Doug s'en alla chez Tymshare, alors que le reste de son équipe fut recrutépar Xerox. Et c'est chez Xerox que Jobs vit la souris. Et il décida que Lisa® en serait dotée.

Après les difficultés de commercialisation rencontrées par Lisa®, Jobs voulut offrir les mêmes concepts à un prix abordable par une masse d'utilisateurs. Cet ordinateur serait aussi portable, utiliserait les développeurs indépendants (comme IBM avait si bien su le faire), et le résultat fut le Macintosh™.

Ce sont ces mêmes concepts que l'on retrouve aujourd'hui dans "TopView®", dans GEM de Digital Research's, chez Atari et sur l'Amiga.

C'est aussi avec un processeur de la puissance du 68000 qu'un tel ensemble matériel/logiciel fut possible.

Et voici donc le Macintosh™, avec à présent 128 Koctets de mémoire morte (Read Only Memory, ou **ROM**), et jusqu'à 1024 Koctets de mémoire vive (Random Access Memory, ou **RAM**).

ATTENTION, ECOLE...

Dans ce livre, nous partons du principe que vous vous êtes déjà un peu familiarisé avec des termes tels que *bits, octets,*

RAM, ROM, processeurs 8/16 et 16/32-bits, etc. Un niveau minimum est le bienvenu, beaucoup de livres de programmation sont déjà sur le marché, et un de plus n'est pas de mise.

Espérons que vous avez déjà programmé en Basic, ne serait-ce qu'*un petit peu* . Par la suite, nous expliquerons pas mal de spécifités propres au "C", au Pascal et à l'assembleur. Ce qui rendra aisée la compréhension des programmes commentés. Nous réviserons plus spécialement les structures dynamiques, les procédures et les fonctions.

TECHNOLOGIES

Voici les spécifications du Macintosh™:

- Processeur **Motorola 68000** processor - lignes de données de **16 bits**, registres de **32-bit**, opérant à 7.8336 Mhz.

- Mémoire morte de **128 Koctets,** mémoire vive de **1024 Koctets (2 x 512) ou 512 Koctets (2 x 256).** Sur cette mémoire, un Koctet est attribué à des buffers (mémoire tampon) de disque et de son, **22 Koctets** sont réservés à la mémoire écran (bit-mapped), et **3 Koctets** sont utilisés pour le buffer du système.

- Un lecteur de disquette interne, dont la capacité est de **400 Koctets (ou 800 sur le Mac +)** formatés, et un taux de transfert de **375 Kbit/s**. La vitesse de rotation est variable, allant de 390 à 605 tours/mn.

- Disquettes de 3 pouces 1/2. 80 ou 160 (double face) pistes ainsi réparties:

12 secteurs sur les 16 pistes externes, 11 secteurs sur les 16 suivantes, et 8 secteurs sur les pistes internes. Ce, afin de conserver une même densité linéaire d'informations.

- Clavier: Apple a voulu faire usage du clavier mécanique traditionnel. Un Intel 8021, doté de sa mémoire gère une **table d'événements** toutes les 3 millisecondes (ou *ms*).

Il faut plus d'une ou 2 ms pour qu'un contact électrique soit

établi à partir d'un contact mécanique. Voilà pourquoi la table d'événements est balayée une seconde fois 5 ms après le premier balayage. Si on a appuyé sur une touche, l'information est transmise via le registre à décalage du 6522 du Macintosh™, au système ou au programme d'applications qui en tiendront éventuellement compte.

- écran à résolution de 512 x 342 pixels.

- Interfaces: RS 422A, RS 232 C, ayant une vitesse de transfert de 230.4 Kbits/sec, et presque 1 Mbit/sec en utilisant une horloge externe.

- SCSI (Small Computer System Interface), appelé "Scuzzy". Il permet des transferts d'informations depuis ou vers des périphériques à des vitesses de 12 Mbits/seconde.

- Souris à action mécanique et optique (phototransistor), 90 points au pouce, pouvant enregistrer des mouvements relatifs.

- Générateur de son: convertisseur analogique/digital, à échantillonage de 22 Khz. La sortie audio vous permet d'utiliser Macintosh™ comme pré-ampli, et de le connecter à un synthétiseur ou une chaîne HI-FI.

- Circuit horloge/calendrier CMOS, alimenté par une pile pouvant durer deux ans.

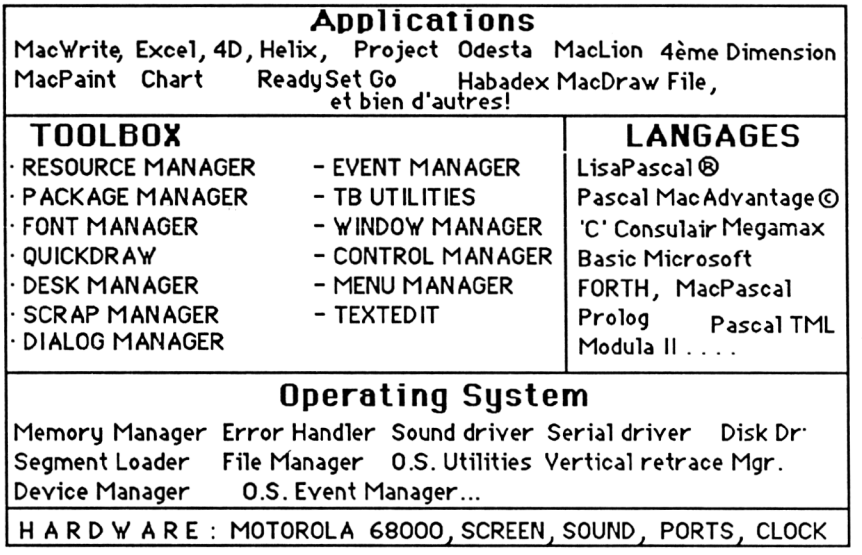
- Circuits PAL (Programmable Array Logic). Huit de ces circuits remplacent 50 autres circuits TTL. Ils assistent le 68000 en ce qui concerne:

  - la génération d'adresses linéaires,
  - la génération de signaux analogiques,
  - séquencement de la mémoire,
  - gestion du bus, etc.

Sur l'IBM PC, l'**interface** couleurs/graphiques (pas la carte-mère, simplement l'interface), comporte 40% de circuits intégrés de plus que dans tout le Macintosh™!

Des appels au gestionnaire de Bureau (DeskTop Manager) suffisent pour rendre les "Accessoires de bureau" capables de s'exécuter en même temps qu'une application.

L'affichage s'effectue sur un écran de 9 pouces utilisant un circuit DMA (Direct Memory Access).



DE MEMOIRE DE MAC...

S'il vous fallait écrire un programme "*à la Mac*" pour un ordinateur qui ne serait pas le Macintosh™, il vous faudrait définir vos propres menus, les graphiques, l'affichage de texte, le contrôle du curseur, etc. Mais vous n'aurez jamais ces problèmes sur le Macintosh™: un ensemble de routines - dont la plupart se trouve en mémoire morte - gère tout ceci à votre place. A vous de les apprendre, et de les appeler.

Définissons, dans un premier temps, des routines comme étant des définitions de **types** de données utilisées dans les procédures et fonctions, ainsi que les procédures et des fonctions.

Une procédure ou une fonction, c'est un ensemble d'instructions écrites par un programmeur qui utilise un langage connu par la machine; le langage est connu par la machine de manière **indirecte**, puisqu'il doit passer par un **compilateur**, ou par un **interpréteur** afin d'être traité par le processeur de l'ordinateur.

Chaque routine a sa fonction propre pour laquelle elle a été écrite.

SECRETS DE SORCIER

Le secret du Macintosh™, c'est précisément sa mémoire morte: toutes ces routines, testées, donc *déboguées*, sont mises "en conserve" dans une mémoire rapide d'accès. Elles utilisent une caractéristique de la famille de processeurs du 68000: les **pseudo-instructions**.

Tout code d'opération débutant par **1Ø1Ø** ("**A**" en Hexadecimal), est traité d'une manière spéciale. Le 68000 traite cette opération comme une "pseudo-instruction", examine les 12 autres bits, et se dirige vers une "dispatch table" (table de re-direction) logée en mémoire basse, qui renvoie le 68000 à la routine existante.

Le 68000 traite donc ce type d'opération comme si elle faisait partie de son propre jeu d'instructions. "Trap Dispatcher", une routine située aussi en mémoire basse, dirige les pseudo-instructions à leur adresse. Mis à part un des processeurs de la gamme de Digital Equipment, nous ne connaissons pas d'autres processeur agissant de même.

L'emplacement de chacune de ces routines en mémoire morte est connu, sous forme de schéma codé dans la ROM. Lors du démarrage, le système lit cette information et la charge dans la "trap dispatch table" qui occupe 1024 octets en mémoire vive.

Sur le Mac +, vous vous apercevrez (dans le chapitre consacré au gestionnaire de mémoire), de la présence de plus d'une table d'indirection.

On peut accèder à cette table via les instructions GetTrapAddress and SetTrapAddress, deux parmi les nombreuses routines de la ROM (mémoire morte). Il est aussi possible de modifier l'adresse d'une routine, et de la faire pointer sur du code que vous pourriez écrire.

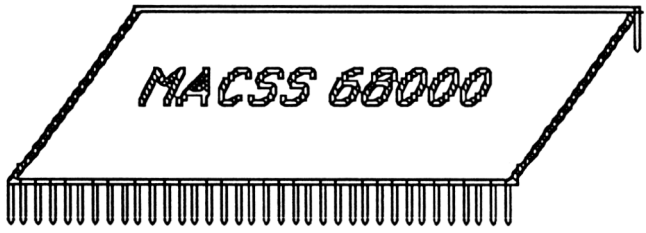
PROFESSION ? ASSEMBLEUR

Dans tout assembleur, et celui du 68000 ne fait pas exception à la règle, on a la possibilité de définir des *noms symboliques*. Ceux-ci peuvent reprendre des adresses

ayant un rôle spécifique, des masques, des déplacements, des valeurs numériques, etc. Par exemple, la variable globale du système nommée **"Ticks"** récupère, exprimée en 1/60ème de seconde, le temps écoulé depuis le dernier chargement du système. On en a besoin pour dater certains événements, certaines opérations sur fichiers, pour le calcul de nombres aléatoires.

On sait où elle se trouve, parce qu'on a défini une zone portant ce nom. Et à cette zone correspond une adresse.

32 QUESTIONS - 200000 REPONSES.



(Tout ce que vouliez savoir sur le 68000 sans oser le demander...)

Le Motorola MC68000 est un processeur puissant, puisqu'il peut transporter sur son dos plus de 68000 transistors. Son fils, le 68020, en transporte plus de 200000.

Le bus de données sur 16 bits donne accès à deux octets, à chaque cycle machine.

Le 68000 contient des aides au déboguage, un mode trace, des "trappes" le protégeant des instructions (ou des adressages) illégales, sept niveaux d'interruption, et deux modes: utilisateur et superviseur.

TYPES DE DONNEES

Le 68000 opère sur cinq types de données différents:

- bit,
- 4 bits Binary Coded Decimal (BCD),
- 8 bits (octet),
- 16 bits (mot),
- 32 bits (long mot, long ou longint).

Le suffixe **.B** suivant l'instruction signifie qu'elle traitera un **opérande** d'un octet de long.

Le suffixe **.W** traiteun **mot**, et **.L**, un double mot (long mot).

Les lignes UDS and LDS (dans le hardware) permettent de couper un mot en deux octets. Une adresse d'octet peut être impaire ($00 0120, $00 0121,...). Les mots et les double-mots seront toujours alignés sur des frontières de mot, donc paires. L'adresse d'un double-mot paire, mais pas forcément divisible par 4.

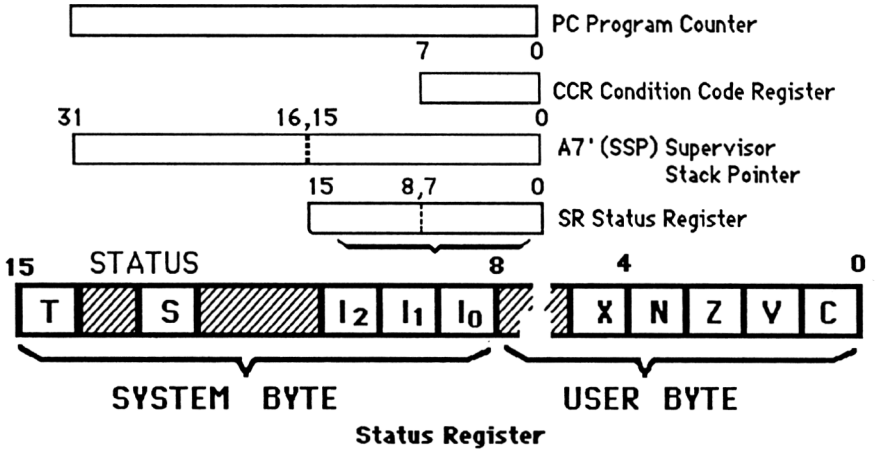
L'alignement est requis lorsqu'on crée des *ressources* en assembleur. Les ressources sont des paramètres. On peut les trouver à l'intérieur d'un programme écrit en assembleur, ou encore dans un fichier des ressources. Des routines de la ROM permettent de l'appeler.

Le 68000 vous permet de spécifier des constantes ou des données quelconques en notation binaire, octale ou hexadécimale. Avec ses 56 types d'instructions sur 5 types de données et ses 14 modes d'adressage, il comporte au total près de 1100 instructions. Le 68020, qui prend la relève, offre plus de 90 types d'instructions. Suit un aperçu des possibilités du 68000.

SIGNEZ LE REGISTRE S.V.P.

On peut aussi utiliser les registres de données comme compteurs, ou index (en général, ils servent à emmagasiner des données).

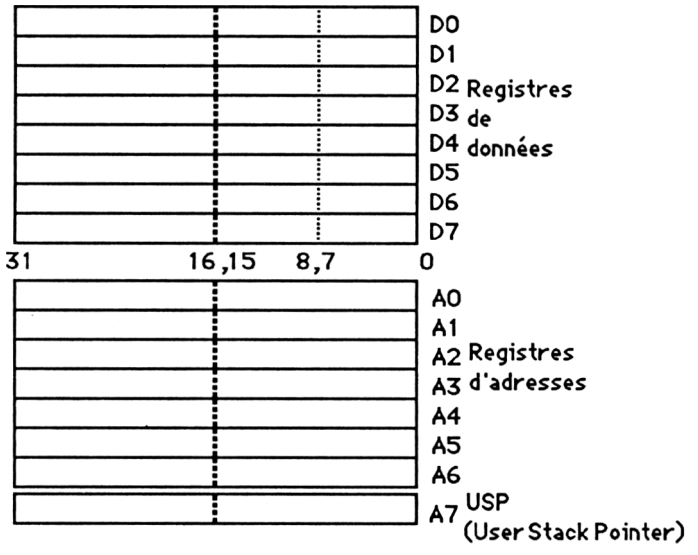
Les registres **adresses** ne descendent pas au niveau de l'octet.



Status Register

Le bit 13 du Status Register (Registre d'état) fait passer le processeur du mode "superviseur" au mode "utilisateur".

Le bit 15 fait passer le processeur en mode "trace".



QUELLE EST VOTRE ADRESSE?

Le registre A7, c'est le pointeur de pile, aussi connu sous les initiales SP (Stack Pointer). Le registre A5 pointe sur le TOS (Top Of Stack), sommet, si l'on peut dire, de la pile. PC signifie 'Program Counter', et SR, Status Register ou "registre d'état".

Vous trouverez ci-dessous, une liste des 14 modes d'addressage (six groupes). Tout ceci s'adresse plus spécifiquement aux programmeurs en langage assembleur.

1) Registre de donnée, direct
   (a) donnée, directe
   ```
   MOVE #'HELLO',DØ  ; place 'HELLO' dans DØ.
   MOVE #256,DØ      ; place la valeur décimale 256
                       dans DØ.
   ```
   (b) addresse, directe
   ```
   MOVE $FEED,A1     ; place la valeur hexadécimale
                       'FEED' à l'adresse contenue
                       dans A1.
   ```
   EA (Effective Address) = Dn ou An ou SP
   Dn: nº du registre de donnée; An: nº du registre d'adresse.
   ```
   MOVE.L  D0,$1F00  ; place le contenu de DØ à
                       l'adresse $1FØØ.
   ```

2) Registre, indirect (en abrégé, r.i.)
   (c) EA = (An):
   ```
   MOVE.W (A3),D1    ; place le contenu de l'adresse
                       contenue dans le registre A3
                       dans D1.
   ```
   (d) registre indirect avec post-incrementation.
   EA = (An); An + 1 ---> An
   ```
   MOVE.L(SP)+,A0    ; retire un long-mot de la pile et
                       le place dans AØ.
   ```
   (e) r.i. avec pré-décrément
   An - 1 ---> An; EA = (An)
   ```
   MOVE.W D3,-(SP)   ; place le mot contenu en D3 sur
                       la pile.
   ```
   (f) r.i. avec déplacement
   EA = (An) + d16
   ```
   ADD.L D0,$4(A1)   ; additionne le contenu de DØ au
                       contenu de l'adresse A1 + 4
                       octets. Le résultat est stocké à la
                       même adresse. .L indique au
   ```

68000 que l'addition porte sur 4 octets.

(g) r.i. avec déplacement et index court:
EA = (An) + (Rx) + d8 . . . . . (Rx = An ou Dn)
ADD.W D0,$6(A1,D2) ; additionne le contenu de DØ à celui de l'adresse A1 + D2 + le déplacement de 6 octets.

(h) r.i. avec déplacement et index long.

3) Absolu
(i) r.i. absolu court (un mot)
(j) r.i. absolu long (sur 24 bits)

4) Relatif au PC (Program Counter: compteur de programme)
(k) Relatif au PC avec déplacement
EA = (PC) + d16
MOVE < label >,D0

(l) Relatif au PC avec index
EA = (PC) + (Rx) + d8. . . . (Rx = An ou Dn)
MOVE < label > (A5),D1 ; index **et** déplacement.
MOVE YANN(A5),D1 ; place le contenu de l'adresse [YANN + A5] en D1.

5) Implicite
(m): implicite réfère à SP (stack pointer), PC (program counter), USP (user SP), SSP (supervisor SP) ou SR (Status Register) sans les nommer. Lorsqu'un BRA s'exécute, (BRanch Always), il implique le compteur de programme (PC).

6) (n) Immédiat

Permet le transfert d'un octet, d'un mot ou d'un double-mot vers un registre ou un emplacement mémoire.

Selon l'instruction, la longueur de l'instruction est comprise entre **un** et **cinq** mots.

L'**octet utilisateur** du Registre d'état contient les bits de Carry, Overflow, Zero, Négatif et Extend. Ce sont des **drapeaux** (flags), qui dépendent du résultat de certaines instructions.

Les bits I0, I1 et I2 de l'octet système sont utilisés comme masques d'interrupt pour gérer les sept niveaux d'interrupts vectorisés.

Pénétrons plus profondément dans l'**architecture** du Macintosh™; tout d'abord, revenons sur cette caractéristique de la famille du 68000, appelée "**instructions non implantées**".

On accède à la ROM via une table en mémoire appelée **Trap Dispatch Table (TDT)**.

Toute instruction débutant par IØIØ, 'A' en hexadécimal, fournie au 68000, est considérée par celui-ci comme une instruction "non-implantée" ("unimplemented" instruction), qui provoque un interrupt. Celui-ci est géré par le microprocesseur, qui renvoie sur une adresse définie par l'utilisateur (Apple pour ses "ROM", ou vous, si vous programmez en assembleur, ou si vous appelez une sous-routine située en ROM). La TDT donne l'adresse de la routine correspondante, qui s'exécute. De cette manière, on peut considérer que les routines sont des **extensions** du jeu d'instructions du 68000.

Dans le Macintosh™, les adresses sont sur 24-bits, permettant 16 Mégaoctets de RAM. Elles sont contenues dans des **longints.** Mais, afin d'être plus "compactes", les adresses de TDT sont écrites sous forme de **MOTS** (16 bits), permettant 512 adresses sur 1 Koctet.

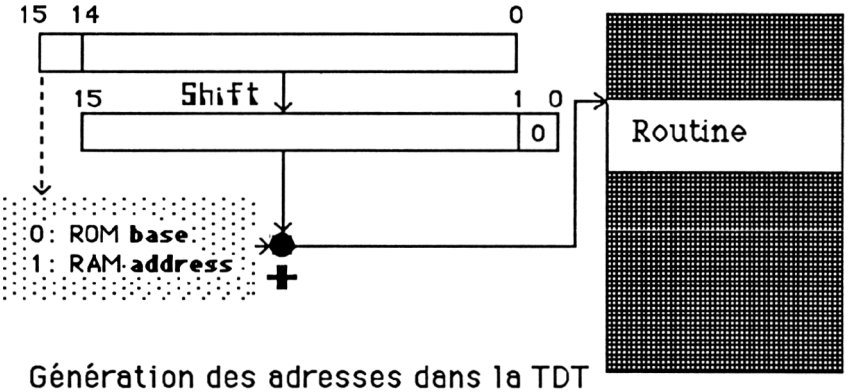
Le bit de poids fort (bit 15) indique si la routine est en ROM (Ø), ou en RAM (1). Les 15 autres bits constituent un déplacement (**offset**) à partir d'une adresse de base. Celle-ci est: soit le début de la ROM (romBase), pour les routines situées en ROM, soit le début du heap système (ramBase), pour les adresses de RAM.

Un déplacement sur 15 bits vous permet d'adresser jusqu'à 32 Kmots. C'est suffisant pour la ROM, mais pas pour la RAM. C'est pourquoi le nom "ramBase" (défini dans les fichiers .EQU de l'assembleur) pointe sur le début du heap système, non sur le début de la mémoire (le Heap est dynamique, le stack est statique), de manière à donner plus d'extension. Il en résulte que:

- Toute routine située sous le heap ne pourra être accédée par la TDT.

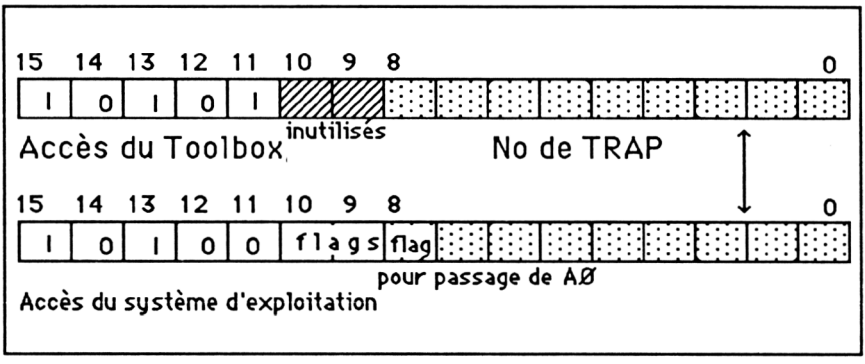
- Le heap pourra éventuellement contenir des adresses inaccessibles à la TDT. Sur le Mac +, qui comporte ses 1024

Koctets de RAM, il fallait trouver une solution: nous la verrons en détaillant le Memory Manager.



Génération des adresses dans la TDT
(Trap Dispatch Table)

Quelques-unes des macro-instructions de l'assembleur génèrent un "trap word" (zone de 16 bits, débutant par IØIØ, et considérée comme une "instruction non implantée). La TDT se sert d'une partie des 12 octets restants comme d'un index. La routine réside soit dans la Toolbox, soit dans l'O.S.



Les macro-instructions débutent par un "_" (par convention, dans le MDS, Mac Development System. Elles sont ainsi aisément repérables), suivies du nom Pascal ou Assembleur. Ce point est particulièrement important, soit pour le programmeur, soit pour celui qui "veut tout savoir sur son Macintosh™", puisque le même nom (à quelques - rares - exceptions près) se retrouve dans TOUS les langages. C'est cohérent, net, clair.

exceptions près) se retrouve dans TOUS les langages. C'est cohérent, net, clair.

Les exemples suivants fournissent du code en assembleur. Voici ce que vous pourriez trouver au début d'une "boucle d'événements", en attente d'un "clic" de la souris ou d'une frappe du clavier:

```
EventLoop                          ; MAIN LOOP

    _SystemTask
    ; PROCEDURE  TEIdle(hTE:TEHandle)
    MOVE.L       TextHReg,-(SP) ; Handle associé au record text
    _TEIdle
```

Voici à présent in exemple (non relié au précédent), d'une fonction AND:

```
.FUNC AandB
MOVE.L (A7)+,A0  ; sauvegarder l'adresse de retour,
                       postincrement.
MOVE.W (A7)+,D0  ; B, un entier, 2 octets, donc un MOT.
MOVE.W (A7)+,D1  ; A, idem.
AND.W  D1,D0     ; devinez
MOVE.W D0,-(A7)  ; (ou -(SP)) résultat passé sur la pile (stack)
                       pour le retour au programme
JMP    (A0)      ; Nous avons stocké l'adresse de retour
                       dans A0
.END             ; Toute bonne chose a une fin!
```

Le programme Pascal qui suit pourrait très bien appeler cette fonction:

```
program Monprog;
    var a,b:integer;
    function AandB(a,b:integer):integer;
    external; { signifie que AandB est externe }
              { au programme }

begin
    a:=254;   { bit 0, bit  de poids le plus faible}
    b:=33;
    Writeln(A,B,' et A and B : ',AandB(A,B));
end.
```

Il faudra passer des paramètres (si la fonction appelée le demande) depuis et vers les routines en ROM via la pile, dans le cas d'une routine "stack-based" (basée sur la pile), comme si elles étaient passées depuis un programme Pascal, ou via un registre du 68000, dans le cas d'une routine "register based" (basée sur un registre).

Il est aussi important de savoir que la plupart des appels en assembleur utilisent la séquence d'appel en Pascal que vous trouvez dans "Inside Macintosh™" (IM). C'est parce que les couches de logiciel du Mac on été conçues en Pascal.

INDIVIDUS TYPES

Le type et la quantité de paramètres, une fois de plus, dépendent de la procédure ou de la fonction appelée. Vous trouverez dans le tableau ci-dessous les contenus de la pile, selon leur type en Pascal:

TYPE Pascal	occup. mémoire	Contenu
integer	2 octets	entier en complément à 2
real	4 octets	bit de signe, expos.: 8 bits et mantisse: 23 bits
longint	4 octets	entier en complément à 2
string[n]	4 octets	pointeur sur la chaîne de caractères; le premier bit pointé indique la longueur de la chaîne de caractères (en char)
record	1 à 4 octets	contient la variable si la taille de celle-ci est ≤ 4 octets; autrement, c'est un pointeur sur la variable
array	comme pour le "record"	

TYPE Pascal	occup. mémoire	Contenu
Pointer	**4 octets**	**adresse de la valeur de la donnée**
Handle	**4 octets**	**adresse du master pointer**
Byte	**2 octets**	**valeur dans l'octet de poids faible**
SignedByte	**2 octets**	**comme pour le "Byte"**
char	**2 octets**	**valeur ASCII dans l'octet de poids faible**
boolean	**2 octets**	**valeur dans le bit Ø de l'octet de poids fort: Ø, false; - 1, true**

Quand au passage de paramètres de type **variable**, c'est toujours l'**adresse** de la variable qui est passée dans la pile. donc, la taille passée est de 4 octets en ce cas, quel que soit son type; en Pascal, nous utiliserions l'opérateur @ pour y accéder, ou l'opérateur **&** en "C" (cf le chapitre suivant, "Pascal").

Si la routine appelée est une fonction, il nous faudra prévoir la place, dans la pile, pour le résultat de cette fonction. Pour ce faire, on "push" le nombre d'octets requis sur le stack (ou encore, on met à zéro le même nombre d'octets). Ensuite, on "push" les paramètres dans la pile, dans l'ordre dans lequel ils arrivent dans la liste de paramètres. Même démarche (à part le résultat de la fonction) pour les procédures.

Le pointeur de pile doit toujours être aligné sur un mot, donc avoir une adresse **paire**. Il y a donc un octet de padding dès qu'un nombre impair d'octets est placé sur la pile. Il n'y a aucun contrôle d'effectué si le SP (stack pointer) est manipulé directement (comme c'est le cas pour quelques instructions).

Le système du Macintosh™ ne se sert pas de la distinction possible des modes utilisateur et superviseur. Tous les programmes, système ou applications, sont exécutés en mode **superviseur**. Ceci implique le fait que la pile de l'application est la même que celle du système. La procédure de passage des paramètres est rigide et ne doit pas être changée, sinon, catastrophe!

DE LA ROUTINE...

Les routines offrent au développeur de logiciels quelques avantages:

Réduction du temps de développement: cela prend du temps de mettre soi-même au point des routines d'affichage, de menus, de création de fenêtres. Et comme sur le Macintosh™, elles vous arrivent toutes faites, testées, vous pouvez être certain qu'elles fonctionnent correctement.

Le seul effort à fournir, c'est au début, quand vous investissez dans leur apprentissage. (bien que des applications, comme c'est le cas sur le Mac, l'Apple II C, l'Amiga ou l'Atari 1040 ST Atari, qui sont "event driven", c.a.d. liées aux événements, soient plus difficiles à déboguer).

En fait, le premier programme que vous pourriez écrire vous servira de **cadre** pour les suivants.

Une interface utilisateur cohérente vous permet de créer des applications immédiatement comprises par l'utilisateur. Ponter-Cliquer, c'est le mode opératoire suffisemment puissant pour réduire de beaucoup les manuels utilisateurs. De plus, ceux-ci ne seront jamais très épais si, en plus, vous avez prévu une "aide" (help) que l'on peut appeler depuis l'application.

La mémoire vive disponible est plus importante: l'utilisation de routines résidant en ROM, dont la taille est hyperoptimisée réduit le code de l'application. Et vous avez aussi la possibilité de segmenter les applications.

La connexion directe des routines appelées au 68000 divise par deux le temps d'exécution de celles-ci (vous rappelez-vous le passage du Mac 512 au Mac + ?).

Nous avons décidé de ne pas traduire "Manager" par "Gestionnaire". Parfois, nous utiliserons ce dernier terme, afin de bien rappeler qu'il s'agit d'un gestionnaire, mais si vous plongez dans "Inside Macintosh™", toujours pas traduit en français, du moins, par les américains, vous serez familiarisé avec le terme. Quant aux "drivers" et "handler", nous les garderons sous la même appellation. Ce ne sont pas des "Managers" à proprement parler, mais il s'en faut de peu: les fonctions et procédures de ces derniers interrogent, ou attribuent des valeurs, à des zones systèmes.

Et voici un premier principe démystificateur:

Les routines du Macintosh™ sont un ensemble de: TYPES de DONNEES, FONCTIONS et PROCEDURES regroupées par fonctionnalité. Chaque fonctionnalité exige son type d'information spécifique. Cette information sera trouvée dans le RECORD correspondant.

Exemple: un ***événement*** doit être reconnu, localisé et daté; c'est pourquoi on trouvera les routines de l'Event Manager basées sur l'**EventRecord**, qui comprendra des **champs** (ou zones) définissant le **type** d'événement, **à quel moment** il s'est produit, et **où** il a eu lieu (sur l'écran). Ce n'est qu'un exemple succinct: certains des champs cités peuvent contenir d'autres informations dépendamment du type.

Voici un premier aperçu (il y en aura d'autres) du jeu de routines du Macintosh:

L'OPERATING SYSTEM, et l'INTERFACE UTILISATEUR

Macintosh Software : an overview	
Basé en RAM soft de haut niveau: package de conversion hexa/binaire Package d'utilitaires internationaux Package standard de gestion de fichiers.	Basé en ROM **TOOLBOX:** **QuickDraw** **Dialog Manager** **Menu Manager** **TextEdit** **Control Manager** **Window Manager** **Tool Event Manager** **Desk Manager** **Scrap Manager** **Package Manager** **Font Manager** **Resource Manager** **Tool Utilities**

soft de bas niveau (système d'exploitation)	
Printer Driver **Printing Manager** **O.S. Event Manager** **Device Manager** **Disk Driver** **Serial Drivers** **Sound Drivers** **File Manager** **Vertical retrace Manager** **System Error Handler** **O.S. utilities.**	**Memory Manager** **Segment Loader** **Floating point package** **Transcendant. funct pack** **Disk initialization pack.** **AppleBus Manager**

DES TRUCS DANS LE SYSTEME.

L'**Operating System**, travaillant au niveau le plus bas, fournit le support nécessaire aux applications, afin qu'elles utilisent le "hardware". Il occupe 1/3 de la ROM, et comprend:

Vertical Retrace Manager:

Soixante fois par seconde, le circuit video du Mac génére un interrupt de trace verticale (vertical retrace interrupt). **TOUT** dans le Mac est basé sur ce pouls du système, aussi bien la compaction de la mémoire, que les ré-affichages.

L' O.S. Event Manager:

L'insertion d'une disquette, la frappe sur le clavier, le mouvement de la souris, le clic de celle-ci, une information provenant d'un réseau, sont autant d'**événements**. Ils sont transmis au système ou à l'application via l'O.S. Event Manager. Certains événements peuvent être générés par certains des Managers eux-mêmes.

L' AppleBus Manager:

Les programmes d'application (en bref: les applications) peuvent envoyer et recevoir des informations au travers d'un

réseau AppleBus. L'AppleBus Manager est chargé de cette tâche.

L'Operating System contient aussi: L'O.S. Core Traps (TRAP DISPATCHER, INTERRUPT HANDLERS, etc.), Keyboard/Mouse Handler (manipulateurs (!) de clavier/souris), les utilitaires O.S., le Serial Driver (série), le Sound Driver (son), le Disk Driver, Device Manager, et le File Manager. Nous reviendrons plus en détail sur tous ces termes.

Le Segment Loader (chargeur de segment):

Il charge du code de programme en mémoire. Ce code se présente soit en un seul bloc, soit sous forme de segments de code chargés dynamiquement, afin d'économiser la place mémoire. Car il est possible d'écrire (ou d'utiliser) un monstre qui réside totalement en mémoire. Mais alors, vous aurez des difficultés si vous voulez utiliser le reste de la mémoire, sous Switcher, par exemple.

Le Memory Manager (Gestionnaire de mémoire):

Il alloue (ou libère) dynamiquement les blocs de mémoire utilisés par les applications. Il minimise la fragmentation de la mémoire en permettant à des blocs d'être **relogeables** (ceux-ci sont toujours accédées de manière indirecte), et fournit un genre de "cache" en purgeant de manière optionnelle - si vous lui dites de le faire - les objets au fur et à mesure que la mémoire se remplit. Si un programme a besoin de plus de mémoire que celle du Mac (ou de la partition allouée, sous Switcher, par exemple), le gestionnaire de mémoire trouvera son chemin parmi les blocs de mémoire ou même, sur l'espace disque.

I/O System: c'est le support pour des gestionnaires de périphériques, basées en RAM, pour les gestionnaires (ici, "pilotes") de disques, et pour les deux ports d'E/S. La plupart des appels E/S sont effectués sous forme **asynchrone**, ce qui permet à des processus internes de s'effectuer alors qu'une opération d'E/S a été lancée. Le File Manager s'occupe des accès fichiers. Le Device Manager gère les E/S physiques.

Le System Error Handler:

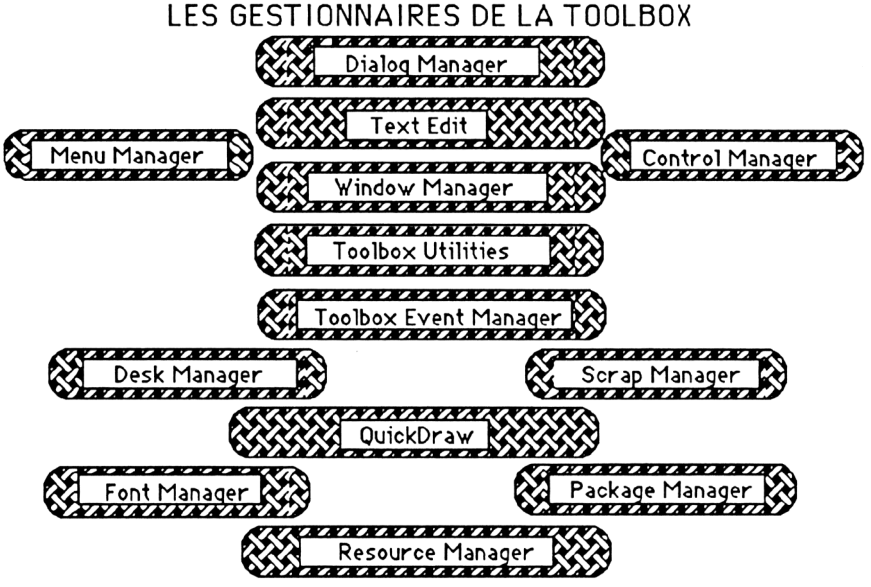
Une "boîte d'alerte" (alert box) vous avertit d'une situation dangereuse, comme une "disquette pleine". Lorsque se présente

une telle situation, l'Error Handler se charge de vous avertir, au cas ou le programme d'application ne la gère pas. Une boîte d'alerte arrête l'application, qui ne repartira que si certains paramètres ont été fournis (en utilisant un clic sur un bouton, ou si vous indiquez que vous êtes au courant de la situation).

Le reste de la ROM est occupé par:

L'User Interface Toolbox
(La boîte à outils de l'interface utilisateur).

GERER LES GESTIONNAIRES DANS LA BOITE A OUTILS



Multiples sont les interactions entre les diverses parties du Toolbox. C'est pourquoi nous plaçons le Ressource Manager au niveau le plus bas.

QuickDraw:

C'est la partie principale du Toolbox. Ecrit en Pascal, pour Lisa, il occupait 160 Koctets. Réécrit en assembleur, optimisé, il tient maintenant en ROM sur 24 Koctets. C'est, tout comme les autres routines de la ROM, un ensemble de définitions de types, de fonctions et de procédures graphiques.

Pratiquement TOUT ce que vous voyez sur l'écran, QuickDraw le dessinne, y compris le texte, avec toutes ses variantes (italiques, gras, etc.), les figures géométriques, remplies ou non de motifs, des zones arbitraires de l'écran (nommées "régions" - ce concept est différent sur l'Amiga). QuickDraw peut appeler d'autres Manager, tel le "Font Manager", s'il s'agit d'afficher du texte.

Les appels à QuickDraw s'affichent à l'intersection de trois régions que nous étudierons. l'Update region (région de "mise à jour" conserve les parties qui ne sont plus visibles, si elles viennent d'être recouvertes par une fenêtre, par exemple). On peut donc faire se recouvrir les fenêtres, et garder traces des modifications.

QuickDraw a aussi cette faculté de pouvoir enregistrer des séquences d'ordres, et de les conserver sous forme de "picture" (dessin). Vous pouvez les rappeler à l'échelle désirée.

Resource Manager:

Le gestionnaire de Ressources contrôle le texte des menus, les icônes, les formes de curseurs, le texte des boîtes de dialogues, les jeux de caractères. Tout ceci est conservé dans un "fichier des ressources", externe au programme - bien que vous puissiez TOUT définir dans un programme assembleur. Mais la philosophie du Macintosh™, c'est de tout garder à l'extérieur, afin de n'avoir pas à recompiler l'application entière en cas de modification d'un de ces textes. C'est le cas pour les traductions de logiciels.

Tout, dans Macintosh, est "ressource", y compris le code. Les ressources, c'est une librairie qui permet au code de pêcher les données dont il a besoin. Le code (les routines du Resource Manager) va chercher les données, un peu comme dans des

boîtes. Chaque boîte est identifiée par le TYPE de ressource (FREF, WDEF, INTL, PICT...) et la ressource de ce type, à l'intérieur de chaque boiite, est identifiée par son numéro.
Ce qui forme un logiciel "géré par table" (table-driven software).

Il vous est possible d'utiliser des ressources pré-définies, ou des ressources que vous définissez vous-même.

Voici quelques exemples de ressources. Dans le chapitre consacré au "Resource Manager", nous noterons les règles de syntaxe auxquelles les ressources doivent obéir, afin d'être compilées par RMaker, le compilateur de ressources.

```
Type MENU
    ,128
     /14
    A propos de MonProg
    (-

    ,129
    Fichier
          (Nouveau
          (Ouvrir
* . . . . . .
```

les voici commentées:

```
* Exemple de ressources; les lignes qui commencent par un '*'
* sont des commentaires
* Yann FAIN, 04/03/86
*
Type MENU
    ,128
*  Nº de Menu
      /14
* Menu Pomme, contenant en général le "A propos de...
* et les accessoires de bureau
    A propos de MonProg...
* première ligne de ce menu
    (-
* ligne de pointillés
* C'est tout pour le menu Pomme. On ne décrit pas
* les accessoires de bureaux à ce niveau

    ,129
* deuxième élément du menu
    Fichier
```

* titre du Menu
(Nouveau
(Ouvrir
* ils se présenteront en grisé la première fois que vous les
* appellerez (par SetMenuBar)
* .....

Menu Manager:

Il contrôle les menus, et les "tiroirs" qui leur sont associés. Il n'y a qu'une seule barre de menus. Vous devez théoriquement comprendre tout nouveau logiciel en explorant ses menus. Même recommandation pour tout logiciel que vous écrivez.

Window Manager:

Une fenêtre, c'est un lieu sacré. Toute information affichée ou saisie, dans le Macintosh™, l'est dans une fenêtre.

Ainsi, on n'écrit pas **directement** sur le bureau.

Le Gestionnaire de fenêtre gère tous les événements liés à une fenêtre: sa création, ses dimensions, la mise à jour de son contenu, et même, une liste chaînée des fenêtres superposées.

Si un utilisateur "clique" dans une fenêtre, le Window Manager sait reconnaître **où** dans la fenêtre s'est produit le clic: dans la barre de menus, dans une barre de défilement (une barre de défilement, c'est un **contrôle**).

Control Manager:

Il gère tout ce qui est lié au dimensionnement des fenêtres, et les autres types de contrôles, comme les "radio-boutons", les boîtes de contrôle (check-box), et les boutons (par exemple: SUITE, ANNULER, entourés d'un rectangle)

Dialog Manager:

Les dialogues sont des fenêtres qui contiennent des éléments (items) tels des contrôles, des textes que l'on peut éventuellement éditer. C'est dans ce type de fenêtre que l'on saisit l'information à passer à l'application. Les "alertes" sont des dialogues d'un type spécial.

Le Dialog Manager gère les saisies d'informations, et l'information elle-même, toujours pars le biais de certaines routines qu'utilise le développeur de logiciels.

Package Manager:

Il permet l'accès aux routines du Mac qui sont basées en RAM. Elles comprennent un ensemble de virgule fixe et virgule flottante, d'utilitaires de récupération de formats de dates, jour, année, de notation utilisée (selon le système installé, exemple: un système en anglais comprendra les jours de la semaine sous forme Sunday, Monday, alors qu'un système français ou québecois récupérera Dimanche, Lundi...). Ces routines peuvent ultérieurement être placées en ROM.

Il y a en fait, dans le Mac cinq packages situés en RAM: un formatteur de disquette, un gestionnaire de fichiers (standard file package), les utilitaires internationaux mentionnés plus haut, les routines en virgule flottante sur 80-bit au standard IEEE 754, et les fonctions transcendentales elles-aussi au standard IEEE.

Les jeux de caractères, les accessoires de bureau comme l'horloge, le puzzle, et la calculette sont aussi dans le fichier système standard.

Utilitaires du ToolBox:

C'est un ensemble d'utilitaires qui gérent les chaînes de caractères, les convertir en longint, effectuer des opérations au niveau du bit, déplacer des blocs de mémoire, etc.

Scrap Manager:

Il gère le transfert de données vers ou depuis le presse-papier, afin de les utiliser dans d'autres applications. Celui-ci peut recevoir des objets de types divers.

C'est grâce à lui qu'on peut passer de l'information d'une application à l'autre, par le "Copier-Coller".

Desk Manager:

Gère les accessoires de bureau, sur disquette, ou en mémoire (Clavier, Tableau de bord, etc.).

TextEdit:

TextEdit contrôle l'écran et l'entrée/sortie des données du clavier, à un niveau très bas.

Font Manager:

Il est ordinairement appelé par QuickDraw. Il effectue le chargement des divers jeux de caractères, et si celui-ci n'est pas en mémoire, il appelle le Resource Manager.

Lorsque la taille d'un jeu de caractères ne correspond pas à celle qui est spécifiée, le Font Manager se charge alors de la mise à l'échelle.

Event Manager:

Nous avons déjà défini ce que sont les événements. L'OS Event Manager les récupère, et les transmet au ToolBox Event Manager.

Celui-ci peut alors les gérer et les passer à l'application. Ce qui entraîne la conclusion suivante: tout logiciel pour le Macintosh™ est une boucle de gestion d'événements.

Mac n'en a pas l'air, mais il est truffé (tout est relatif!) de processeurs qui assistent le 68000: un SCC (contrôleur de communications série) Zilog 8530, et un Intel 8021 contrôlant le clavier.

Ainsi, vous avez l'impression que les tâches s'effectuent en même temps, et vous pouvez entrer du texte de la main gauche, tout en bougeant la souris de la main droite, et vous voyez les résultats à l'écran.

EN PARLANT AU FINDER

Hors de la ROM (point de salut), on trouve le Finder. C'est en soi une **application**. C'est le système d'exploitation du Macintosh™. Il peut lancer les applications, organiser les fichiers, les renommer, les effacer, les copier, les dupliquer. Il peut gérer les objets graphiques (icônes) liés aux fichiers ou aux dossiers, aussi bien que le texte.

La métaphore du Bureau, c'est la base de la philosophie Mac. TOUT se trouve sur le Bureau. Rien n'est en-dessous. Visicalc (et ceux qui sont venus ensuite, tel Multiplan et Lotus) apportait le principe de "feuille électronique". Le Mac apporte celui de "Bureau électronique". Les fichiers, les applications, sont des "objets" "posés" sur le Bureau.

Le Finder n'a pas de "directory" arborescente (la directory, c'est un fichier des fichiers). En revanche, il peut répartir les données dans des "dossiers" (folders).

Le Finder est donc une application, et il se sert d'un fichier nommé le "System". Comme leurs icônes sont identiques, on ne s'en rend pas compte. Le système contient des données et du code, ressources sur lesquelles nous reviendrons, et qui sont des *paramètres* pour le Finder. Le Finder, lui aussi, a ses propres ressources.

La directory de chaque disquette réside en RAM, et reste en mémoire, même si une autre disquette est chargée par la suite, même si elles ne se trouve plus *physiquement* dans le lecteur.

C'est pourquoi vous pouvez examiner le contenu d'une

disquette dont l'icône est sur l'écran, alors que la disquette est déjà rangée.

Le Finder gère aussi les échanges de disquettes. Chaque application relie une icône particulière au(x) document(s) qu'elle crée. Il y a même une icône standard et un document standard, pour les applications qui n'ont pas défini d'icône particulière.

Chaque fichier, chaque document, est relié à une application par la **signature** de son **créateur**. Son **type** vient compléter cette information, et permet à un fichier d'être utilisé par toute la famille d'applications qui sait comment elle est structurée.

Un document issu de MacPaint, par exemple, peut être utilisé par l'Album (Une mini-application du Menu Pomme), pour être transféré sur MacWrite.

Lors de telles opérations, le Finder se sert de l'horloge, pour dater un ensemble d'informations, comme la création ou la modification d'un fichier, quand on assigne des labels de blocs de données (qui sont utilisés pour rebâtir une directory, au cas où).

Nous avons évoqué les ressources du fichier System. Celles-ci peuvent être modifiées, comme, par exemple, des assignations de touches, les formats de date, les identifiants demonnaies, etc.

TOUT TEXTE apparaissant dans un programme peut être modifié par l'Editeur de ressources, s'il a été défini comme une ressource. C'est donc une tâche aisée de **personnaliser** ou **traduire** un programme, de rajouter des "Option-Lettre" là où vous en voulez.

Pour ceux qui écrivent le logiciel, il leur suffit de changer le texte dans les ressources, de recompiler en utilisant le compilateur de ressources (RMaker), et de procéder au "Link" du logiciel. Celui-ci, dans sa phase "objet", n'aura pas été modifié et voilà,

un *programme en français!*

Ouf! C'était donc notre première approche du tas de gestionnaires, packages, événements que l'on rencontre dans le Mac. Vous méritez bien une bonne nuit de repos.

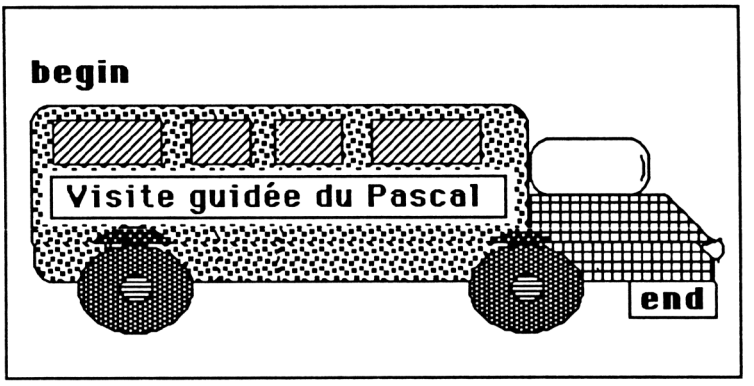
Demain, vous recevrez votre **épée**:

Pascal

CHAPiTRE II

BLAISE ET VOUS:..

La visite guidée du Pascal.



PASCAL-IFORNIE

Pascal, ce sera le langage de référence dont nous nous servirons, car on le comprend (presque) comme une langue vivante. Dans ce chapitre, nous réviserons quelques points fondamentaux. Ils sont essentiels, et un (léger) effort est requis de votre part, si vous voulez comprendre les programmes commentés. Il faut le lire, sinon, la garantie de votre Mac saute...

Quant au "C", disponible sur le Mac, nous en mentionnerons certaines caractéristiques. Il vous sera ainsi plus familier, et vous comprendrez aisément les lignes de code écrites en ce langage.

Aujourd'hui, on trouve sur le marché plusieurs compilateurs Pascal, qui permettent de développer des logiciels. Pour les adresses de fournisseurs, sautez à l'Appendice A.

On trouve par exemple le Pascal de Think Technologies (ceux qui ont créé le MacPascal, vous savez, l'interpréteur de Pascal, qui peut être à la rigueur utilisé pour apprendre ce langage). Ce sont eux qui ont lancé le MacPascal, interprété.

On trouve aussi le Pascal TML, de Tom Leonard (Eau Gallie, Melbourne, Florida). Il a beaucoup de **restrictions** par rapport au LisaPascal, et il est plutôt lent, quand on travaille sur disquette - moins en RamDisk, évidemment. Le Link est ultra rapide, et sélectif (ce qui augmente la rapidité du link, et réduit la taille du code, car il n'inclut pas l'intégralité des librairies). Il n'accepte pas le **sizeof** dans les déclarations, n'admet pas les définitions de constantes après une définition de type, et ne compacte pas les *packed array of boolean* comme le fait le LisaPascal.

Ainsi, les déclarations suivantes:

```
Type
Rec1  =     record
   Champ1 : integer;
   Champ2 : packed array[1..8] of string;
   Champ3 : longint;
        end;

Const
Long_rec1= sizeof(Rec1);
```

sont possibles en LisaPascal, mais **pas en Pascal TML**.

Il existe aussi le compilateur de SofTech, que nous mentionnons sans jamais l'avoir vu ni essayé.

Et enfin, le Pascal de Step Lively (On Stage, Latham, NY, exit 6 après Albany sur l'Interstate 87 Nord) qui est censé compiler très rapidement, offrir des **UNITS** (portions de code déjà écrites et implantées dans une librairie commune) séparées, un débogueur symbolique, censé aussi s'écrire comme du LisaPascal, avec toutes ses libertés par rapport aux normes ISO ou UCSD. Dont on nous dit que le temps de compilation est hypersonique, le code résultant aussi épais qu'un portefeuille après le passage de l'inspecteur des impôts, mais que personne n'a jamais vu (Michel Andrew Rynderman nous a téléphoné trois

ou quatre fois pour nous promettre la version développeur et ce, depuis milieu Aout 1985!).

Et enfin, le Pascal du système de développement Apple, qui doit toujours venir incessamment sous peu. En bref, et de très loin, c'est encore le LisaPascal qui se trouve être LE langage de développement des français.

Les américains se sont mis au "C", et là, par contre, on trouve de très bons compilateurs: Mégamax, Aztec, Consulair, et celui, plus récent, de Think Technologies (encore eux). Sous Workshop, on trouve aussi un "C". Pas essayé, pas de commentaires.

En utilisant un pré-processeur "C", on peut éventuellement écrire du code qui ressemble un peu au Pascal:

```
/* définitions de types */
typedef     char              Char     /* ! */
typedef     short             Integer
typedef     unsigned char     Boolean
typedef     long              LongInt
typedef     char              SignedByte  /* un octet */
typedef     unsigned char     Byte     /* pour le FontMgr */
typedef     SignedByte        * Ptr    /* pointeur "aveugle" */
```

Enfin, nous avons souligné, dans le chapitre précédent, l'importance, et l'omniprésence des routines du Mac.

Un logiciel Mac sera constitué, en moyenne, de 40% de langage, et de 60% d'appels aux routines.

Quel que soit le langage dans lequel une application est écrite (sauf Basic), on utilisera toujours **FindWindow** pour savoir dans quelle partie d'une fenêtre un utilisateur a cliqué.

La partie *code* étant réduite s'estompent les différences entre langages (exception faite pour le Forth).

QUEL EST VOTRE NOM ?

Les noms, ou identifiants, servent à distinguer entre elles les variables, et/ou les constantes, les types, les procédures et les fonctions. En LisaPascal, seuls, les huit premiers caractères sont significatifs. Attention, donc: on se fait insulter par le compilateur, et on ne sait pas pourquoi!

Ci-dessous les trois types de données rencontrés en Pascal:

- simple,
- structuré, et
- pointeur.

Mais Pascal permet de créer d'autres structures à partir de types de données simples pré-définis. Certaines de ces structures peuvent être plutôt abstraites. Voici les types simples de données: (reportez-vous au chapitre I pour leurs longueurs respectives en Pascal):

integer (entier, mot) : -32768,0,32767
longint (double-mot) : -2147483648,0,2147483647
boolean (booléen) : false, true (faux, vrai)
char : ASCII

Ce sont des types "ordinaux". Sur ces types, on peut appliquer les opérateurs suivants:

- ORD : "numéro de série"
- PRED : prédécesseur
- SUCC : successeur

real : réel, allant de 0,±1.4E-45,±3.4E38, au standard IEEE.
string : chaîne de caractères

Voici quelques structures simples:

- array (table, ou tableau, ou matrice),
- record (ensemble de champs, ou de zones, *struct* en "C"),
- set (ensemble),
- file (fichier).

Un tableau comporte un **nombre fixe d'éléments**, tous du **même** type. Le type est défini lors de la déclaration du tableau.

Ex: Matable: packed **array**[1..6] of char;

Un record est une structure comportant un certain nombre d'éléments. Nous en verrons de nombreux exemples.

Un ensemble définit un certain nombre de valeurs de type ordinal. On ne peut donc définir un "set of real".

Quant aux pointeurs, nous les aborderons en détail dans ce chapitre, vu leur importance.

PROCEDURIERS

Pascal permet aussi de créer:

- des procédures,
- des fonctions,
- et des blocs.

Un bloc est constitué de deux parties:

- les déclarations,

- et les instructions.

Les identifiants déclarés dans un bloc sont "locaux", et ne peuvent être utilisés hors du bloc dans lequel ils ont été définis. On peut cependant attribuer les mêmes noms à d'autres variables. Ce ne sont **pas** les mêmes variables d'un bloc à un autre.

En Pascal, un identifiant doit être déclaré (nom, type), avant d'être utilisé.

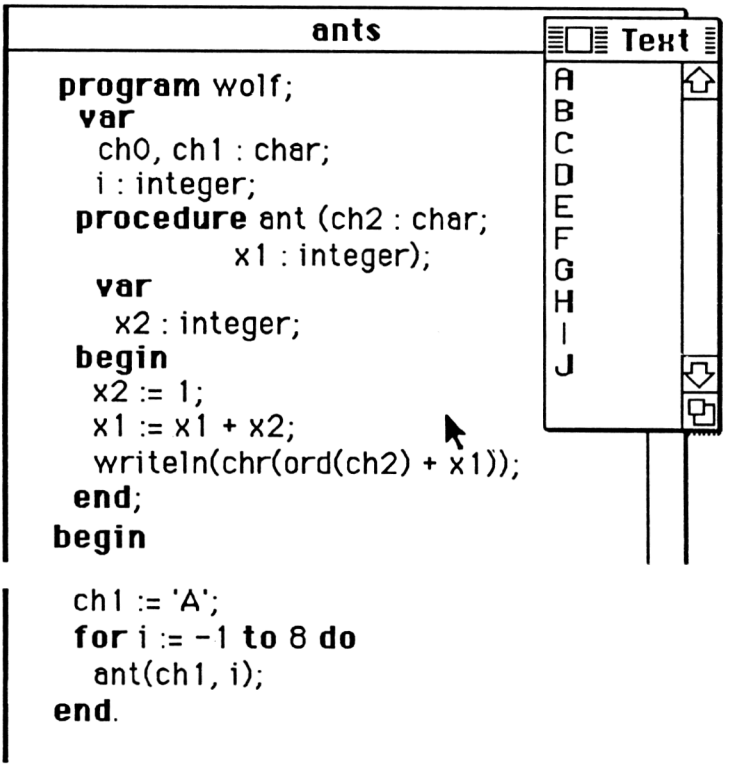
La seule exception à la règle, c'est le pointeur:

```
Unptr=^MonRec;
MonRec=record;
          I,J:integer;
```

```
        Suiv:UnPtr
    end;
```

La section suivante vous apportera davantage d'explications sur les pointeurs.

Considérons le programme suivant, écrit en MacPascal:



ch0, ch1 sont des variables **globales**, x2 est locale à la procédure "ant". Dans la déclaration de la procédure "ant", ch2 est un paramètre **formel passé par valeur**. Et si l'on voit:

function ant(**var** x2: char):**real**;

x2 est un **paramètre formel passé par référence**; le résultat de la fonction sera un *réel*.

Lorsqu'on appelle une procédure, on peut lui passer un, deux, plusieurs, ou même aucun paramètre. Dans le programme ci-dessus, ch1 est le paramètre **actuel** passé à la procédure.

L'intérêt des procédures c'est de n'avoir pas à répéter les mêmes lignes de code lorsque des tâches similaires se présentent.

VOUS ETES FICHE...

Le type **file** définit une collection d'objets externes au programme. Ils sont tous du même type, et un seul d'entre eux est accessible à un moment donné.

C'est le *buffer de fichier* qui donne accès à cet élément du fichier.

Dans un fichier de quotations en Bourse, on pourrait trouver:

```
type

    valeurs=real;                    { valeurs à considérer }
    j_ouv=(LU,MA,ME,JE,VE);          { jours ouvrables }

    status=(ouvert, haut, bas, clot); { type énuméré }
        { quotations d'ouverture, la plus haute, }
        { la plus basse, et de fermeture }
    quotations=record
        jour:array[j_ouv] of
                record
                 date:string[10];
                 prix:array[status] of valeurs
                end;
            end;

var
    sto_quot:    file of quotations;
    CeJour:j_ouv;     { jours ouvrables }
    LeStatus:status;
    Unevaleur:valeurs;
```

Pour utiliser **date**, vous pouvez spécifier la dénomination complète:

```
sto_quot^.Jour[j_ouv].date := '30 JUIN';
```

ou, plus simplement, en se servant du "**with**":

```
with sto_quot^ do <instruction>;
     Jour[CeJour].date
```

ou encore, puisque cette notation n'est intéressante qu'avec plus d'une instruction, dans le sens où elle permet d'économiser sur le nombre de caractères frappés:

```
with sto_quot^ do
     begin
          Jour[CeJour].date := '30 JUIN';
          Jour[CeJour].prix[haut] := 24832.25;
     end;
```

On peut utiliser des "with" à l'intérieur d'autres "with".

Un autre exemple illustrera:

1/ comment avoir accès à de telles variables,

2/ comment indexer une structure de tableau au moyen de scalaires, LU, MA, ME..., en utilisant les **type** et **var** définis ci-dessus:

```
with sto_quot^ do
     begin
          for Jour:=LU to VE do
          begin
             readln(Jour[CeJour].date);
             for LeStatus:=ouvert to clot do
                  readln(Jour[CeJour].prix[LeStatus]);
          end;
     end;
```

Dans cet exemple, nous avons entré les prix allant de **ouvert** à **clot**.

Pour utiliser une construction identique en "C", il faut d'abord définir les **scalaires**:

```
#define LU   1
#define MA   2
. . . . . .
#define TRUE  1
. . . . . .
```

Nous nous devons de mentionner ici le fameux **CASE...OF**, qui accepte la clause **OTHERWISE** dans la version 3.0 (et la 3.9) de l'environnement de développement qu'est le Workshop. Cett clause n'existait pas sur l'Apple II™ ni sur l'Apple III™, ni même sur le Workshop 2.0.

Il sert à répartir des actions, en fonction de valeurs diverses d'une même variable. Son équivalent Basic serait, avec la variable entière *chose* :

ON chose GOTO 1000, 2000, 3000;

En "C", le CASE..OF est légèrement différent; il est précédé d'un "switch", puis, est répété chaque fois, ce qui donne:

```
if (GetNextEvent(everyEvent,&myEvent) == TRUE)

        switch(MyEvent.What); {

        case UpdateEvt:
             SetPort(theWindow);
             BeginUpdate(theWindow);
             TEUpdate(&theWindow->portRect,hTE);
             EndUpdate(theWindow);

        case ActivateEvt:
             instructions... }
```

En Pascal, on trouve aussi, les **records** variants (en "C", des Unions), les boucles **FOR...DO**, **REPEAT...UNTIL**, et **WHILE...DO**: toutes ces composantes de Pascal seront explicitées au fur et à mesure de leur usage.

Rappelons aussi les **UNITS** (actuellement, spécifiques au Workshop), que nous avons déjà mentionnées. Il est aussi possible de créer ses propres UNITS.

DES POINTEURS, POINTES, ET LEURS "^ENFANTS"

Nous le savons déjà, Pascal peut manipuler des **pointeurs**. Avec les fichiers, on les appelle des "types de données dynamiques", puisque le compilateur ne peut associer aucune adresse spécifique aux objets de ces variables. De la mémoire est allouée pour ces objets au moment de l'exécution, et non au moment de la compilation.

Un pointeur, c'est une indication de l'*adresse* d'un objet quelconque. Un pointeur sur ma maison, ce n'est pas moi, mais mon adresse et moi sommes liés: c'est l'endroit où l'on peut me trouver. Simplement, si je déménage, mon adresse devra être mise à jour. Et sur Macintosh™, on déménage beaucoup!

C'est pourquoi, on trouvera des pointeurs de pointeurs, des **handle**.

On crée un pointeur en lui donnant un nom et en lui associant un type, celui de l'*objet pointé* (c'est à dire de l'objet vers lequel pointera le pointeur).

Un pointeur est associé à une variable ayant le même **type de données** que celle qui est spécifiée dans sa déclaration:

```
type
     ptr_list=^piece_detachee;
     piece_detachee =
          RECORD { symbole souligné permis }
            pd_id:integer;
            pd_nom:string[30];
            pd_prix:real;
            pd_ptrsuiv:ptr_list;
          END;

var
     MonPointeur: ptr_list;
```

MonPointeur pointe à présent sur **piece_detachee**.

En utilisant la déclaration ci-dessus, on peut utiliser **pd_nom** de la manière suivante (*pd*, initiales de **p**iece_**d**etachee, et *nom* comme identifiant de champ. Il est plus facile d'utiliser pd_nom pour nommer le libellé d'une pièce détachée dans un fichier stocks, que *truc*, *toto*, *chose* ou *machin*):

MonPointeur^.pd_nom sans avoir à référencer piece_detachee

Dans une instruction "read" : **read**(MonPointeur^.pd_nom);

Dans une instruction d'assignation:

MonPointeur^.pd_ prix:=30.4;

Et si MonPointeur pointe sur un **record** piece_detachee, nous pouvons accéder au prix (pd_prix) de l'enregistrement piece_detachee **suivant** de la manière suivante:

MonPointeur^.pd_ptrsuiv^.pd_prix := 30.4;

Et si pd_ptrsuiv avait été le premier champ de piece_detachee, il aurait été possible d'écrire:

MonPointeur^^.pd_prix := 30.4;

Ce point est important à rappeler chaque fois qu'une structure de type *record* contient elle-même un record, comme nous le verrons bientôt.
Avec les pointeurs, on peut utiliser la notation "**with**" comme avec les noms de fichiers, tout comme dans l'exemple de *with* fourni prédémment.

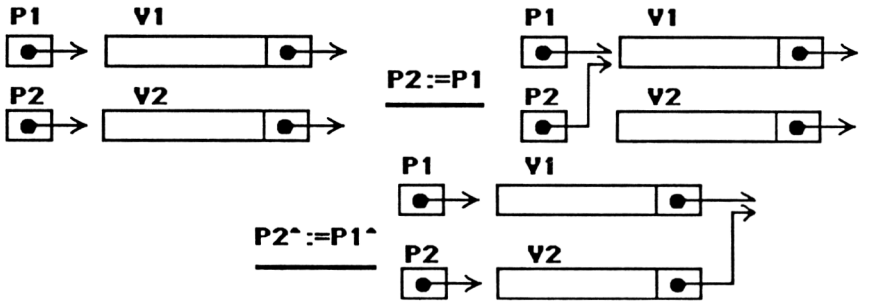
En fait, définir un pointeur a deux effets. Le premier est de **relier** deux éléments (le pointeur, et l'objet sur lequel il pointe). Le second est de dire au compilateur le nombre d'octets à réserver pour la zone pointée.

Voici quelques autres points concernant les pointeurs (!):

```
type
    UneChose=Chose;
    UnPointeur=^Chose;
    ..................

var
    MonPtr, TonPtr:Unpointeur;

begin
    ..................
```

```
MonPtr:=TonPtr;   { MonPtr adresse alors le même objet }
                  { que TonPtr }

MonPtr^:=TonPtr^; { rend l'objet sur lequel pointe MonPtr }
                  { identique à celui sur lequel pointe TonPtr }
```

Dans ce dernier exemple, V1 est égal à V2, et le pointeur contenu dans chaque *record* pointe sur la même *adresse*, sur le même *objet*.



Si on passe un paramètre par référence:

f(**var** i:integer; j:real):integer;

c'est son *adresse* qui est passée à la fonction via le stack. Voyons maintenant quelques exemples de passage et calcul d'adresses. Ces opérations comprennent l'opérateur @, la compatibilité de types, et la compatibilité d'assignation (et certains compilateurs ne se gênent guère pour vous le rappeler!).

TYPES DE TYPES

Dans les expressions, et dans la plupart des contextes (comme, par exemple, un passage de paramètres à une procédure ou une fonction), la compatibilité de type est requise.

On dit que deux types sont compatibles si:

• ils sont identiques.

• l'un est un sous-ensemble de l'autre, ou les deux sont des sous-ensembles d'un même type.

• les deux sont des types *ensemble*, et leurs types de base sont compatibles (il y a de la récursion dans cette phrase).

• tous deux sont des "string" - chaînes de caractères, même s'ils n'ont pas la même longueur.

On doit expliquer la première assertion, "même type": on dit que deux variables sont de *type identique* si:

- on utilise le même identifiant de type pour déclarer les deux, que ce soit des types simples ou composés

```
type
     UnType=....;

var
     var1: UnType;
     var2: UnType;
```

- on les définit dans la même déclaration:

```
var1, var2:^UnType;
```

Mais:

```
var1:^UnType;
var2:^UnType;
```

ne rend pas les deux variables var1 et var2 identiques. Pourquoi?

^UnType n'est pas un identifiant de type, mais un type défini par l'utilisateur (discrimination!).

Et que pensez-vous des définitions suivantes?

```
type
   UnPointeur=^UnType;
   ........................
var
   var1: UnPointeur;
   var2: UnPointeur;
```

HALLELUJAH, HAENDEL !

Dans le Macintosh™, la mémoire est réorganisée tous les 1/60 sec. Cela veut dire qu'il faudra constamment courir après vos données dans le "heap" (tas de données).

Mais comme nous sommes tous très paresseux, on fera appel à un détective privé, Monsieur **Master Pointer** (ou encore, ***mp***, pour les intimes) pour suivre les données. Et lui fera son rapport à son patron, Mr. Handle. C'est lui qui vous dira où sont vos données quand vous voulez y accéder.

En fait, rien n'est aussi simple: vous pouvez malgré tout utiliser des pointeurs, et créerez des blocs **non-relogeables**. Ils sont responsables de la *fragmentation du heap*.

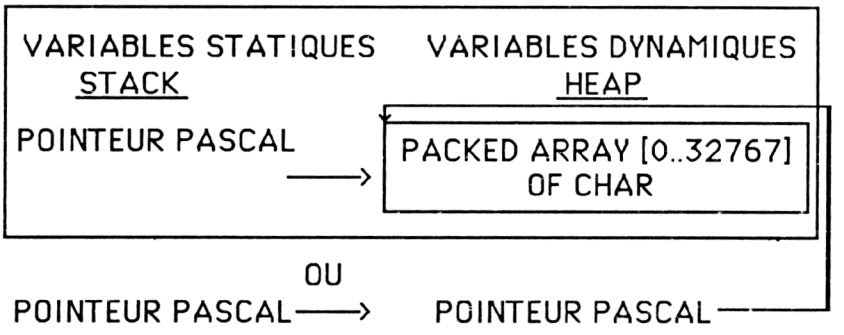
En bref: **Handle -> Master Pointer -> Vos données**.

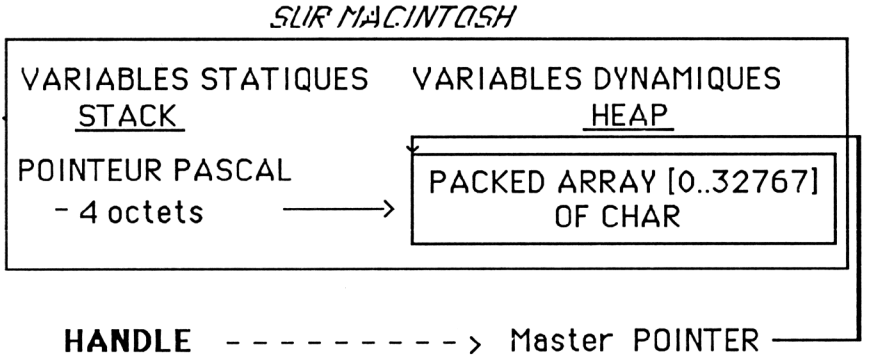
Vous venez de deviner que les **handles** et les **master pointers**, qui sont des données comme vous et moi (!?), sont protégés du compactage qui s'effectue tous les 1/60 seconde.

Les **handle** doivent être définis par **NewHandle** - voir ci-après, et au chapitre consacré au Memory Manager - et, en assembleur, ils doivent être placés dans des variables globales.

Les schémas suivants montrent différentes variables, et comment y avoir accès.

Dans la plupart des ordinateurs:





Attention:

- on peut utiliser des double-pointeurs (voir 'détypage')

- un Handle est un longint. (4 octets)

Après la définition de certains types vient celle du *handle*:

```
UnsignedByte= 0..255;
SignedByte = -128..127; { la plus petite unité que l'on puisse
                          adresser }

Ptr= ^SignedByte;       {ou = ^Nimportequoi; en fait, un
                         pointeur doit, avec le détypage,
                         pouvoir adresser n'importe quel
                         type de données }
Handle= ^Ptr;
.....................................................

MonHandle := NewHandle(SizeOf(UnType));
```

NewHandle(SizeOf(UnType)) crée un bloc **relogeable** dans le "heap". Le *heap* est l'endroit de la mémoire où sont entreposées les données. Cette notion est spécifique aux langages Pascal et "C".

La fonction **SizeOf** retourne la taille d'un objet. Le compilateur ira toujours chercher la compatibilité de type.

ON DEMANDE: ORD4, POINTER, @

Etant donné les définitions suivantes:

```
type
    LeChose     = ^integer;  { pointe sur un entier }
    LaChose     = ^real;     { pointe sur un réel }
var
    UnPtr           :   Ptr; {  rappelez-vous la définition
                               du Ptr }
                        { UnPtr pointe sur SignedByte }
    UnChose         :   LeChose;
                        { LeChose est ici défini comme }
                        { un pointeur sur un entier }

    AutreChose      :   LaChose; { pointeur sur un réel }
```

puisque le détypage (en anglais: "type coercion", ou encore "typecasting" spécifique au langage "C") peut poser des problèmes, on s'en tire bien en utilisant les expressions suivantes:

UnChose:= **POINTER**(**ORD4**(UnPtr));

ORD (ou ordinal, ou ORD4), est le rang d'un objet dans son type.

ORD4(xxx) retourne un longint (32 bits) - une adresse, sur Macintosh™, est codée sur 4 octets, bien que n'ayant que 24 bits.

POINTER transforme un longint en un objet de type pointeur du **même** type que celui de la variable à lauqelle il est assigné. L'argument de cette fonction est un longint repreesentant une adresse physique qui doit être convertie en une variable de type *pointeur.*

En d'autres termes, **ORD4** convertit un type ordinal ou un pointeur et retourne un **longint**.

Un champ récepteur de type longint peut contenir le résultat d'une telle assignation:

ORD4(a)*b;

et **POINTER** convertit un **integer** ou un **longint** en un type pointeur, compatible avec **tout** type pointeur.

Reprenons l'exemple

```
UnChose:= POINTER(ORD4(UnPtr));
```

Une version plus lisible de conversion de type serait la suivante:

```
UnChose:= LeChose(UnPtr); { nous reviendrons sur ce sujet }
                          { mais étudiez cependant cet exemple }
```

@Objet : Cet opérateur permet à un argument de servir de pointeur...sur lui-même, retournant sa propre adresse. Comme l'opérateur "&" en "C". L'adresse qui est retournée peut éventuellement être passée à un autre bloc, ou comme paramètre à une routine en assembleur. L'opérande peut être una paramètre, une procédure ou une fonction.

Dans l'exemple suivant:

```
type
    T1  =packed array[0..31] of 0..1; { 4 octets, ou }
    huit_demioctets = packed array[0..7] of 0..15;

    DeuxOct = packed array[0..1] of char; { 2 octets }
var
    Un_L_Int      :longint;
    regist        :^T1;
    Un_Int        :integer;
    Ptr_ DeuxOct :^ DeuxOct;
```

L'instruction:

```
regist    :=   @Un_L_Int;
```

fait pointer **regist** sur **Un_L_Int**. A présent, **regist^** redéfinit la valeur de Un_L_Int, **comme si** ce champ de 4 octets était un packed array[0..31] of 0..1, et:

```
Ptr_ DeuxOct :=@Un_Int;
```

fait pointer Ptr_ DeuxOct sur Un_Int. Comme vous l'avez deviné, Ptr_ DeuxOct^ **redéfinit** Un_Int **comme si** ce champ de 2 octets de long était un **packed array[0..1] of char**.

Une manière d'examiner l'octet de poids fort (dans un **integer**) et celui de poids faible:

```
type
    i:integer;
    DeuxOctets:packed array[0..1] of 0..255;
```

est de: **moveright(i**, DeuxOctets,2); { pas tous les Pascal }

Note: pour des découpages multiples, vous pouvez aussi vous servir d'un **record** dit "variant". En utilisant des constructions comme celle indiquée ci-dessous, vous pourrez effectuer des "peek" et des "poke" Pascal:

```
function peek(UneAdresse:longint):integer;

    type    UnLMot   = packed array[0..3] of 0..255;
            DeuxZones =   record
                        case boolean of
                          true  : (LAdresse:longint);
                          false : (contenu : ^UnLMot )
                          end;
    var
            peeknpoke : DeuxZones;

begin
    peeknpoke. LAdresse := UneAdresse;
    peek := peeknpoke.contenu^[0];
end;
```

Dans cet exemple, nous avons pu accéder aux quatre octets séparément., comme octets non signés.

Tous ces petits tours sont possibles en LisaPascal. En Pascal TML, en MacPascal, certains packed arrays ont des longueurs totalement différentes, notamment les packed array of boolean.

Avec les record *variants*, on peut même convertir des données d'un type en un autre:

```
procedure Convert_Hexa(n : integer);

    { n est passé à Convert_Hexa }
    { cette procédure écrira n sous forme hexadécimale }

    var
        i:integer;

        hexadecim:packed array[0..15] of char;
```

```
        botharea: record
                    case boolean of
                  true:(x:integer); { 16-bit  }
                  false:(hex:packed array[1..4] of 0..15])
                                { 4 champs de 4 bits chacun }
                   end; { botharea }

begin
    hexadecim:='0123456789ABCDEF';

    with botharea do

        begin
            x:=n; { paramètre passé par valeur }
            for i:= 4 downto 1 do
                write(hexadecim[hex[i]]);
    { on utilise la valeur de chaque champ de 4 bits pour
      indexer "hexadecim" }

        end; { with }

end; { Convert_Hexa }
```

Une fois de plus, toute routine placée en ROM, dans le Mac, est une routine écrite en assembleur.

Si un paramètre est passé par référence, c'est son **adresse** qui est passée à la routine (via le stack):

f(*var* i : integer; j : real) : integer;

Un point important. Si:

nom : **packed array[1..maxchar] of char**;

et *nom* appartient à un record accédé par un pointeur p1, il est possible d'écrire p1^.nom[0], p1^.nom[3], pour avoir accès à chaque caractère de *nom* séparément.

MON COMPILATEUR? UN DETYPEUR!

Même après tout ce que nous avons pu écrire sur la compatibilité de type, les compilateurs sont assez sympa à ce sujet: si une procédure a besoin d'un paramètre de type **T1**, et que vous voulez passer une variable var2, de type **T2,** il vous suffit de **remplacer** var2 par T1(var2) et ainsi, var2 sera considéré comme de type T1. Voilà! Le "C" fait de même.

Répétons l'exemple fourni plus haut:

```
UnChose     :=   POINTER(ORD4(AutreChose));
AutreChose  :=   POINTER(ORD4(UnChose));
{ conversion en utilisant POINTER et ORD4 }

{ ci-dessous, manière plus simple d'écrire en utilisant les
types }

UnChose     :=   LeChose(AutreChose );
AutreChose  :=   LaChose (UnChose);
```

Encore un exemple:

```
type
     TEPtr = ^TERec;
     TEHandle = ^TEPtr;
     Handle=^Ptr;
     .....
var
     TH:TEHandle;
     H=Handle;
     .....
begin
     .....
     H:=Handle(TH);
     .....
     TH:=TEHandle(H);
     .....
```

{ TEHandle est devenu un handle, et H est à présent un handle sur un TERec. Afin de vous familier avec des termes barbares }

Pascal est un langage fortement typé. On ne peut mélanger des fraises avec des pommes, et donc des pointeurs sur des fraises avec des pointeurs sur des pommes! Nous reviendrons sur ces points, en étudiant la mémoire.

L'opérateur @ génère un pointeur du type de NIL, ce qui signifie que ce pointeur est compatible avec tout autre type de pointeur, en ce qui concerne les assignations (:=).

Pour faire pointer **Unchar** sur le même objet que **MonAdr** (un pointeur), il suffit d'écrire:

Unchar:=@MonAdr^;

Unchar est à présent un **pointeur** sur l'objet vers lequel pointe MonAdr.

MACPASCAL

Si vous êtes programmeur novice, procurez-vous un outil qui vous aidera à apprendre. L'interpréteur MacPascal fera l'affaire.

Quels avantages offre-t-il?

- D'abord, il est interprété: vous pouvez exécuter le programme dès que vous l'avez écrit. Tout comme le Basic. La syntaxe de chacune des instructions que vous avez écrite est vérifiée. Ainsi, vous n'êtes pas obligé d'attendre la fin du processus compilation/link pour savoir si vous avez oublié de déclarer une variable, ou si un mot est mal oooortografié, ou une instruction mal écrite.

- Si vous le spécifiez, une vérification de l'ensemble du programme est effectuée,

- Le programme peut appeler la plupart des routines de QuickDraw,

- Vous pouvez travailler sur plusieurs fenêtres: OBSERVE, DRAW, TEXT I/O, et INSTANT (où vous pouvez tester séparément une instruction),

- Les instructions et les mots réservés s'affichent en caractères **gras**, les instructions erronées, en *italique*,

- Vous pouvez suivre aisément la logique grâce au mode pas-à-pas. Avec lui, on apprend vite ce que sont les listes chaînées, et la récursion. Le rêve des profs d'informatique,

- Un compilateur est prévu, et la version "développeurs" est sortie.

Voici un petit exemple écrit en MacPascal:

```
program porto;

procedure flip; (* lecture récursive *)
  var
     c : char;

begin
     read(c);
     if not eoln then
          flip;
     write(c)
end;

begin   (* bloc d'action *)
     writeln('Frappez une ligne, et frappez RETURN');
     writeln;
     flip
end.
```

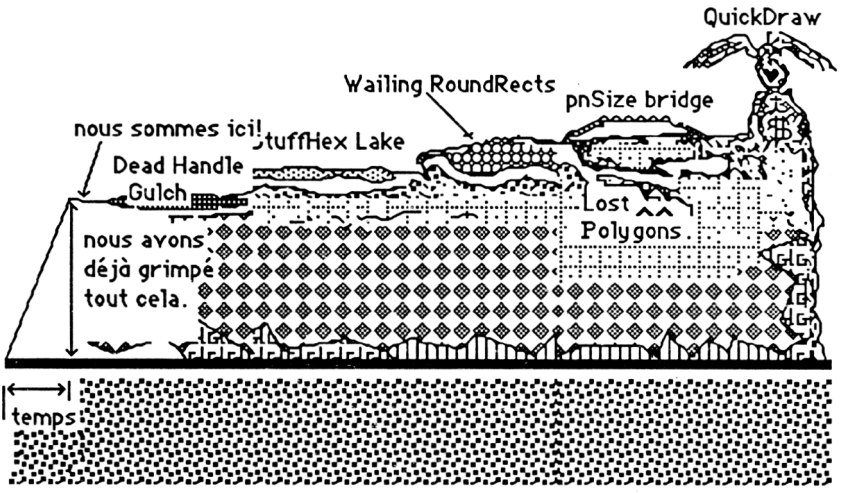
Cet exemple de lecture récursive inverse les caractères lors de l'affichage.

Nous terminons ici notre visite guidée du Pascal, mais nous insisterons sur certains points, au fur et à mesure que nous les rencontrerons. A présent, voici quelques explications sur les caractéristiques principales du Macintosh™.

CHAPITRE III

Camp de base Nº 1

repos, nourriture, Sherpas.



REVUE

Espérons que vous en êtes conscient: nous avons encore du chemin à faire avant de pouvoir comprendre le Mac, et les programmes.

Nous avons posé sur la table un certains nombres d'éléments, de faits, expliquons-les.

Mais attention, il faudra voir **tout**, ou presque, dans le détail avant d'espérer la lumière.

COMMENT ET POURQUOI DES GESTIONNAIRES

Les gens d'Apple ont écrit un ensemble de procédures et de fonctions, de types de données, pour obtenir un ensemble **modulaire**. Ils ont constamment utilisés les **mêmes** définitions de types et de variables, donc, les mêmes **noms**, comme, par exemple, les types **Point**, **Rect**, **cursor** ou **bitmap**, de QuickDraw.

En ajoutant quelques fonctions de base, ils ont créé un **noyau** capable d'effectuer des tâches spécialisées de bas-niveau.

C'est ce noyau qu'on a appelé les... **Managers**. Si vous entrer dans cet ensemble, vous retiendrez vite les fonctions et les paramètres de chacun des **mnémoniques**. Et à quel Manager chacun d'entre eux appartient. La routine '**GetNextEvent**' appartient-elle au Font Manager? Certainement pas.

Ces mnémoniques indiquent, de par les termes, préfixes ou suffixes utilisés, à quel Manager appartient la fonction ou la procédure que vous lisez - ou écrivez. Et dans la "vie courante", pas plus de 100 de ces routines sont **très** usitées. Vous les retiendrez vite.

Comment et Pourquoi une BOUCLE D'EVENEMENTS

Nous avons déjà souligné le fait, un programme Macintosh™ n'est pas un processus rigide et figé, ou un cheminement dans une arborescence.

C'est une suite **d'événements** de divers **types**: le clic de la souris dans la "GoAway" box (boîte de fermeture), ou dans la "scrollbox" (boîte de dimensionnement de la fenêtre), dans du texte (point d'insertion, qui peut faire que le programme modifie la forme du curseur), ou dans la barre de menus, la frappe de texte, la souris maintenue appuyée, l'insertion d'une disquette, un événement de type "réseau", etc.

Toute **interaction** entre l'application, l'utilisateur et le système passe par l'exploration d'une *file* d'attente (First In First

Out) d'événements (*events queue*).

Vous n'êtes cependant pas obligé de traiter les événements dans la séquence dans laquelle ils se sont produits, donc la structure FIFO n'est que théorique. Vous pouvez même en éliminer, en utilisant un *masque* de sélection d'événements. Chaque type d'événement a son propre code numérique correspondant à un bit du masque. Le chapitre consacré à l'Event Manager vous en dira plus long à ce sujet.

Un événement est décrit dans un **record** (EventRecord). Celui-ci contient divers éléments, tel son *type* (Event.WHAT), quand il s'est produit (Event.WHEN); cette mesure est exprimée en **ticks** (1/60è seconde), et les ticks commencent depuis le dernier chargement du système. Il contient aussi le **point** où s'est produit l'événement (Event.WHERE), en coordonnées **globales** à l'écran (et non locales à un GrafPort). L'information contenue dans un EventRecord est aussi bien **qualitative** (Event.What) que **quantitative**.

On trouve aussi des événements avalés et traités par le système, ne requérant aucune action de la part du développeur.

Avec un tel système qui génère des événements au *hasard* (lisez: "au bon vouloir de l'utilisateur"), vous ne pouvez prévoir l'arrivée d'un événement, ni lui réserver une chambre dans le Hilton le plus proche.

Donc, les logiciels pour le Mac devront constamment *tester* une **boucle d'événements**. Celle-ci testera uniquement les événements que le développeur jugera bon d'examiner.

La **boucle d'événements** est le moyen le plus simple d'écrire des programmes et de réduire le code. Une étude des compilateurs et des automates tabulaires montre que c'est la meilleure méthode pour écrire du code compact qui prévoit tout (événements aléatoires). Elle montre aussi que la boucle d'événements est plus facile à tester (les *case..of* se comportent particulièrement bien, très optimisés. L'utilisation de booléens de contrôle aussi, d'ailleurs.

Un automate a des événements de transition et des états. Dans le Mac, les événements générés par le système sont des événements de transition. Les états seront gérés par l'application, via les booléens ou les case..of (ce qui revient au même, puisque le case..of correspond logiquement à un test, et tout test positionne un booléen).

COMMENT ET POURQUOI DES RESSOURCES

En ce qui concerne les Ressources, le choix est naturel: depuis le début, le Macintosh™ a été conçu comme une machine internationale.

Les Ressources permettent de placer la partie **modifiable** hors du code: si vous désirez modifier le texte affiché sur l'écran (un message d'une boîte de dialogue ou d'alerte, un Menu, ou rajouter une touche Commande-x), c'est facile. Modifiez le texte du type STR# dans le fichier des ressources, ou le type MENU, et recompilez les Ressources, non le programme lui-même. Puis, effectuez la liaison des ressources et du code. Notez qu'on n'aura pas eu dans ces opérations à toucher au code lui-même.

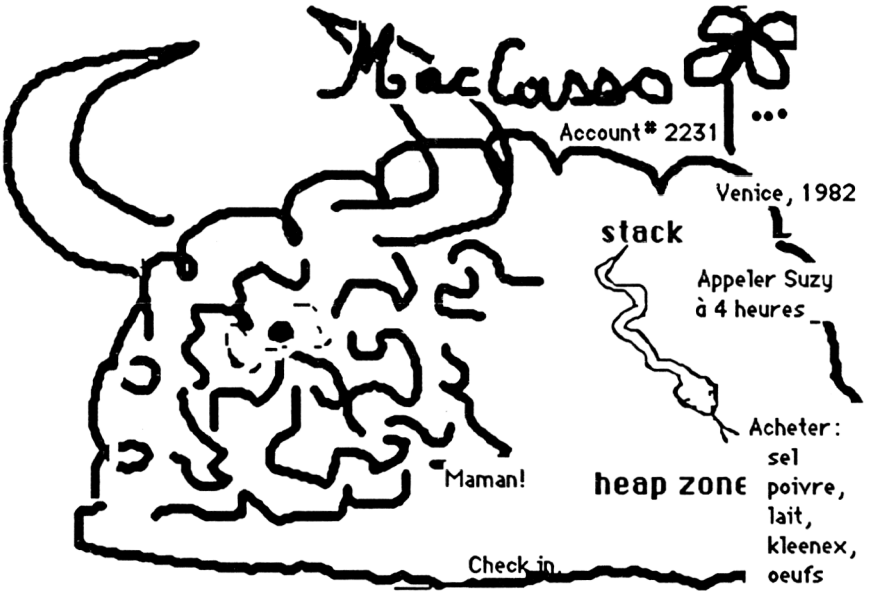
Cette façon de procéder n'est pas nouvelle. Il nous est arrivé de procéder ainsi (sur de grosse machines, en Cobol, PL/1 ou en assembleur. Ce mode opératoire n'est donc pas lié à un langage). Mais la nouveauté, c'est d'avoir bâti la machine **autour** de ce concept.

On dit d'un tel type de logiciels qu'il est "table-driven" (piloté par une table), car tous les paramètres - et des chaînes de caractères correspondant à des messages sont des paramètres à part entière - résident **hors du code**. Celui-ci va chercher les éléments dont il a besoin au moment où il en a besoin, et nettoie (purge) la mémoire dès qu'ils ne sont plus utiles.

Ce minuscule chapitre résume l'ensemble de connaissances du premier niveau. Descendons au deuxième sous-sol en attaquant les Managers. Nous commencerons par lequel, déjà? Ah, oui! Le gestionnaire de mémoire.

CHAPITRE IV

MES MEMOIRES...



EMPILEZ-MOI CE HEAP, S.V.P. MERCI

Nous avons décidé de commencer par le Gestionnaire de mémoire, car la gestion de mémoire apparaît pratiquement partout dans le Mac. Et nous en profiterons pour explorer le Segment Loader. Le Gestionnaire de mémoire ne fait pas partie de la Toolbox, il appartient à l'OS. Est venu le temps d'expliquer ce que sont ces objets nommés "heaps" et "stacks" qui vivent dans la mémoire.

Le stack (**LIFO,** last in first out, c'est la pile). Contrôlée par le 68000, elle est "statique", part du haut et va vers le bas**. Le haut est fixe**, et demain, on enlève le bas. Le bas peut croître ou décroître. Le dernier élément empilé (pushed) est le premier à partir (popped). On n'a aucune action sur le stack, autre qu'y placer ou y retirer des éléments.

La notion de "stack" est liée à la manière d'opérer des microprocesseurs, pas au langage. En Pascal, toute opération sur le stack est gérée par le compilateur.

Sur le 68000, le pointeur de pile, c'est le registre A7. Aussi appelé SP, celui-ci est implicitement appelé par certaines instructions: JSR (**J**ump to **S**ub**R**outine), BSR (**B**ranch to **S**ub**R**outine), et RTS (**R**e**T**urn from **S**ubroutine). Les Registres A5 and A6 sont ici impliqués, puisque A5, c'est le TOS (top of stack), base des paramètres du programme - variables globales -, et A6 est un registre de contexte. Cependant, nous éclipserons cette partie, puisque nous avons décidé de "parler Pascal".

Nous avons déjà mentionné le fait que Macintosh™ travaille exclusivement en mode superviseur (bien qu'il puisse fonctionner en mode utilisateur, avec son registre A'7, User Stack Pointer). Une application partagera donc son stack avec le système. Il en résulte un passage de paramètres rigide. La pile, de par sa structure LIFO, ne permet pas de "trous" (gaps).

Nous avons mentionné le stack, mais il n'est pas directement concerné par le Gestionnaire de Mémoire (vous verrez ce que sont les ID 28 - stack meets heap). C'est plutôt le heap qui nous intéresse.

HAPPEZ CETTE PILE, QUOI!

Le **heap, contrôlé par le Memory Manager** (**MM**) part du bas et croît vers le haut. La notion de "heap" est liée aux langages Pascal et "C".

Le **Heap**, c'est une **portion de mémoire contenant des blocs relogeables, non-relogeables, et libres**. Mais alors, qu'est-ce qu'un bloc?

Les variables dynamiques sont gérées, dans le heap, par des pointeurs:

```
type

    TableDeChar=    packed array[1..26] of char;
    Marker      =   ^TableDeChar;
.....................................................
var

    MaTable:    Marker;       { TableDeChar }
    i       :   integer;
.....................................................
begin
.....................................................
    new( MaTable );
    for i :=1 to 26 do
        MaTable^[i]:= chr(i+64); { ce n'est qu'un exemple! }

    { Note importante }
    { Le pointeur est assimilé au type sur lequel il pointe }
    { Par la suite, vous verrez qu'on donne souvent à }
    { un handle ou à un pointeur le nom du type d'objet }
    { auxquels ils réfèrent }
.....................................................
```

On a ici créé un bloc non relogeable, puisqu'on s'est servi d'un pointeur. Lorsqu'on crée une telle variable, un bloc de taille identique apparaît dans le heap, et le pointeur contient l'adresse physique du bloc (un longint). Puisqu'on peut allouer et libérer de telles variables, vous pouvez avoir plein de place en mémoire, dans le heap, mais peutêtre pas assez d'octets **contigus** pour allouer un bloc important.

C'est ce qu'on appelle la *fragmentation* du heap. La tâche du Gestionnaire de mémoire, ce sera donc de garder le heap aussi compact que possible. Voici comment il procède:

Lorsque le heap se forme (en début d'application), deux zones distinctes sont créées:

Une est **fixe**, l'autre est **relogeable**.

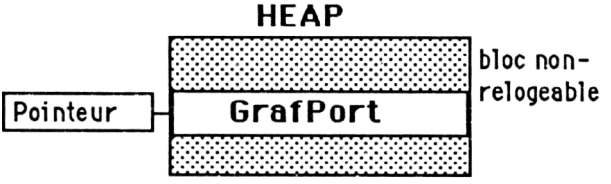
La partie relogeable contiendra les **blocs**. La partie fixe contiendra les **"master pointers"** (**mp** en abrégé).

La longueur de chaque mp est de quatre octets pour chacun des blocs. Au moment du compactage, le **MM** placera les blocs

relogeable, donc relogés, à des emplacements mémoire contigüs. Il mettra aussi à jour les **mp**.

Ceux-ci ne sont pas déplacés et donc, les **handle** qui pointent sur eux ne seront pas affectés.

Les mp sont situés dans le heap, dans sa portion inférieure. Un bloc de mp en contient 64, et sa longueur est donc de 256 octets. Si vous avez besoin de plus de mp, vous utiliserez la procéedure **MoreMasters** (nous l'illustrerons dans les programmes). MoreMasters allouera de l'espace en tranches de 256 octets. Mais c'est en début de programme qu'il faut y penser.

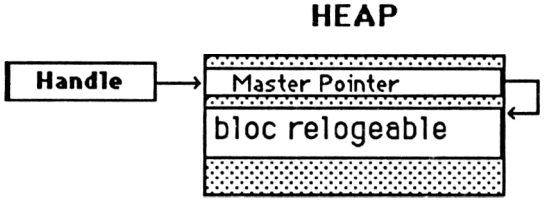


Nous avons vu qu'en créant des variables au moyen de pointeurs, on obtient des blocs **non-relogeables**. Ils ne bougent pas, une fois créés. Nous savons aussi (mais il est important de le rappeler), qu'ils provoquent la fragmentation du heap. Les fenêtres et les dialogues en sont des exemples.

Les blocs relogeables sont affectés par le compactage de la mémoire qui a lieu tous les 1/60 sec. On les manipule grâce aux **handle**. Ils ne causent pas la fragmentation de la mémoire.

Les **Menus**, les **contrôles**, les **segments de code** sont contenus dans des blocs relogeables. Tous ces éléments sont des "ressources", et vous en déduisez que tout ce qui est ressource sera lié au Gestionnaire de Mémoire.

Chaque bloc de heap est accédé de manière indirecte. Il est composé d'un **en-tête**, et du **contenu** du bloc proprement dit. Il contient aussi des octets inutilisés.

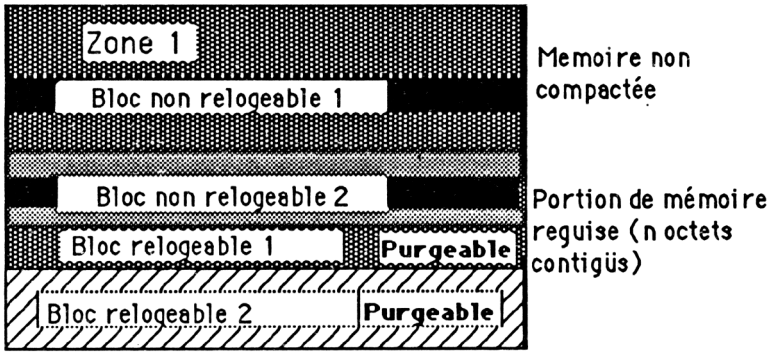


Si vous désirez opérer sur un bloc relogeable, il vous faudra, avant toute chose, le "verrouiller". Lorsque vous aurez fini de l'utiliser, vous le déverrouillerez.

Un bloc déverrouillé est purgeable, ou non-purgeable. Si vous spécifiez un bloc purgeable, il disparaîtra lorsque davantage de mémoire sera requise par le MM, et le handle qui lui est associé contiendra alors NIL.

La pile et le heap partagent la même mémoire. S'ils entrent en collision, cela pose de sérieux problèmes. Les routines du MM vous permettent d'agir sur le heap.

Une de ces routines vous permet de fixer la limite supérieure du heap d'une **application** (ApplLimit). La taille de celui-ci est, par défaut, de 8 Koctets.



Fragmentation du Heap

Si le Heap rencontre le Stack ,

KA-BOOM!

Le MM gère **au moins** deux zones distinctes:

- L'une, initialisée lors du chargement du système, c'est le **heap du système**. Il occupe 46 Koctets (il occupait 16,5 Koctets sur le Mac 128 K).

- L'autre, c'est le **heap de l'application**. Le code de l'application réside dans cette zone, dans une portion de mémoire réservée par le Segment Loader. Une application peut créer différents

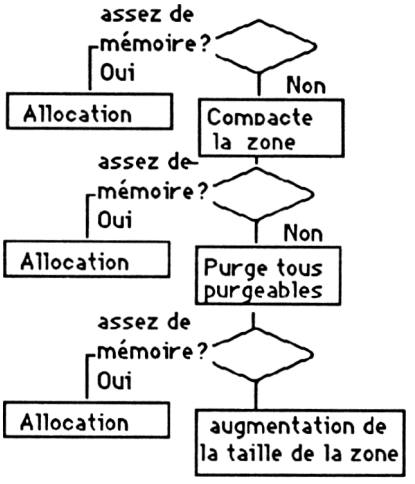
heaps.

Et le Resource Manager a besoin de place pour y placer ses ouailles. Celles-ci sont soit des ressources de l'application, soit des ressources provenant du système.

La plupart des routines du MM s'appliquent implicitement à la zone de heap "courante", disponible à tout moment (lon se sert des routines *GetZone* et *setZone* pour modifier et accéder à **theZone**, en-tête de zone de heap). Le système sauvegarde les valeurs du heap zone courant lorsqu'il doit utiliser son propre heap (system heap).

Ensuite, il restaure ces valeurs. Comme vous avez pu vous en rendre compte, on utilise indifféremment les termes **heap**, **heap zone** ou, plus simplement, **zone**.

Allocation d'un bloc relogeable



L'allocation d'un bloc non-relogeable s'effectue beaucoup plus simplement:

si "Pas assez de mémoire" alors, compactage, puis, accroissement de la zone. Si nécessaire, le MM purge des blocs purgeables.

DEREFERENCER UN(E) HANDLE

On peut avoir accès à un bloc (relogeable, s'entend) de deux manières différentes: par son **mp**, ou par son **handle**. En utilisant le handle, vous écririez:

LeHandle^^.UnChamp, or

LeHandle^^[0] pour accéder au premier octet d'un **packed array** [0..32767] of char;

```
. . . . . .
with LeHandle^^ do
    begin
. . . . . .
```

On nomme cette technique la **double indirection'**. Notez bien qu'un handle accède à un bloc par double indirection, et qu'un pointeur accède au bloc par simple indirection. Mais accéder à un bloc par double indirection requiert une référence supplémentaire. Cela semble inutile, puisque le **mp** retrouve seul son chemin. Alors?

Efficacité oblige, il est parfois préférable de **déréférencer** le handle. Cela signifie qu'on fait une **copie** du **mp** du bloc, dont on se sert pour adresser chaque champ par simple indirection.

A présent, le fait d'utiliser un handle implique le fait que le bloc est relogeable, voire purgeable. Si vous travaillez sur un tel bloc en utilisant une copie du mp, et que ledit bloc est purgé ou déplacé, le mp sera correctement mis à jour. Mais pas sa copie, un banal pointeur, de l'espèce de pointeur la plus commune.

Vous verrouillerez donc le bloc avant de faire une copiue de son mp. En utilisant la fonction HLock. Alors, et seulement alors, vous **déréférencerez** le handle. En fin de chapitre, nous montrerons un autre moyen de **déréférencer** un handle.

```
var
    UnPointer   :   Ptr;
    UnHandle    :   Handle;
begin
    . . . . . .;
    UnHandle    := NewHandle(LaTaille); { on vient de }
                       { créer un bloc relogeable }
```

```
.........;
HLock(UnHandle);                    { verrouillage du }
                                    { handle, et }

UnPointer:= UnHandle^;              { on déréférence le handle }
                                    { en le copiant }
.........;
.........;
...UnPointer^.UnChamp...            { ici, on utilise la simple }
                                    { indirection avec un }
                                    { pointeur  simple }
.........;
.........;

HUnlock(UnHandle);                  { si le bloc n'est pas }
                                    { déverrouillé lorsqu'on a }
                                    { fini de l'utiliser, il provoque }
                                    { la fragmentation du heap }

DisposHandle(UnHandle);             { libère le bloc, le handle, }
                                    { et le mp associé }
.........;
```

Quand vous aurez **vérrouillé** votre handle. vous pourrez utiliser la notation **WITH**:

dans une procédure, vous accédez à un champ comme ci-dessous:

```
MaProc(MonHandle^^.MonChamp);
```

Ou, si MonHandle^^ est un **record**:

```
with MonHandle^^ do
```

ou encore, en utilisant une copie du handle:

```
WITH MonPointer^ do
        begin
                ...
```

Le compilateur Pascal lui-même déréférence les handles.

Il se peut que vous vouliez utiliser une structure dans un bloc verrouillé. Si cette structure contient elle-même une structure, il vous faudra verrouiller les deux avant de les déréférencer.

Il nous faut noter que si les handle sont pratiques en ce qui concerne la gestion compacte de la mémoire, les blocs relogeables utilisent davantage de place que les non-relogeables. De plus, il faut plus de temps pour y accéder si on y accède par double-indirection.

STRUCTURE DE BLOCS ET MASTER POINTERS

Le MM, nous l'avons vu, considère le heap comme un ensemble de blocs, soit alloués, soit libres.

L'en-tête de bloc occupe 8 octets (et se divise en deux fois quatre octets). Il nous dit si le bloc est libre ou alloué, s'il est relogeable ou non. Toute cette information est conservée dans le "tag byte" (ou drapeau), l'octet de gauche du premier long mot. Si les bits 6 et 7 du tag byte (30 and 31 du long-mot) sont à:

- **'00'**, le bloc est libre,
- **'01'**, le bloc est non relogeable,
- **'10'** le bloc est relogeable.

La longueur des octets inutilisés est conservée dans les bits 0 à 4 du tag byte (ou 24 à 27 du long-mot). Cette longueur sert aux corrections de taille pendant l'alignement.

Les bits 0 à 23 du premier long-mot contiennent la taille physique du bloc (en octets).

Si le bloc est relogeable, le second long-mot est alors un pointeur sur le *mp* du bloc, sous forme de déplacement depuis une adresse de base, le début du heap.

Si le bloc n'est pas relogeable, le second long-mot est un pointeur sur la zone du bloc. Dans le cas où le bloc est libre, l'information du long-mot ne sert à rien. Un bloc libre est toujours relogeable, déverrouillé et purgeable.

Le MM ne s'intéresse pas au contenu du bloc. Juste à son en-tête.

Dans les bits 0 à 23 d'un **mp**, on trouve l'adresse du **contenu** du bloc auquel il réfère. Et, on trouve dans les bits:

- 29, le bit de ressource,
- 30, l'indication que le bloc est purgeable (1) ou non,
- 31, si le bloc est verrouillé (1) ou non.

Le bit de ressource permet au Resource Manager de savoir si le bloc en question contient une information qui a trait aux ressources de l'application.

UN PEU PLUS DE GESTIONNAIRE DE MEMOIRE...

Si vous recevez un nouveau logiciel, vous apprenez beaucoup simplement en explorant les menus. S'il est conforme aux principes du Macintosh™ User Interface, il comprend certaines caractéristiques que vous rencontrerez dans tout logiciel:

Fichier, Ouvrir, Fermer, Imprimer, Quitter;

Edition, Couper, Copier, Coller, etc. et vous trouverez probablement les mêmes raccourcis (c.a.d. Command/lettre). Béni soit le développeur du logiciel, s'il a inclus un help que vous pouvez afficher!

Il en est de même en ce qui concerne les éléments trouvés dans les Managers:

Les noms utilisés dans les routines, nous l'avons déjà signalé, informent a priori sur le Manager concerné, et le rôle de la routine. Ceci en préambule aux premières définitions de type, aux premières procédures et fonctions que nous attaquons à présent.

```
const
    maxSize     =   $800000;     { immense };

type
            { les définitions ci-dessous font partie }
            { des 'MemTypes', types de base servant à }
            { construire d'autres types }
```

```
SignedByte   = -128..127;  { n'importe quel octet }
                           { en mémoire }
Byte         = 0..255; { octet non signé pour Font Mgr }

Ptr          = ^SignedByte;    { blind pointer, }
          { pointeur "aveugle", sur n'importe quoi a priori }
Handle       = ^Ptr;           { pointeur sur un mp }

ProcPtr      = Ptr;     { pointeur sur une procédure }
Fixed        = LongInt;     { type arithmétique de }
{ virgule fixe. Plus rapide qu'un type réel  }


Str255       =  String[255]; { taille maximale }
                          { d'une chaîne de caractères }
StringPtr    = ^Str255;    { pointeur sur Str255 }
StringHandle = ^StringPtr; { handle correspondant }

Size         =    longint;

             { Voici quelques types plus abstraits, ne
               faisant pas partie des MemTypes }

PtrSurChose =    ^Chose;
Chose       =    record
                 ......
                 end;

HandleSurChose =    ^PtrSurChose;

var
UnPtrSurChose   :    PtrSurChose;
UnPtr           :    Ptr; { voyez la définition du Ptr }
                  { UnPtr pointe sur un SignedByte }

UnHandleSurChose:  HandleSurChose;
```

RE-DETYPAGE...

Dans les définitions proposées, UnPtrSurChose est un pointeur sur Chose, et UnPtr est un pointeur sur un signed byte. Une Chose, n'importe quoi, peut avoir une longueur différente

de celle d'un signed byte.

Si vous voulez que UnPtrSurChose pointe sur le **même objet** qu'UnPtr, vous pouvez écrire (voir chapitre II):

```
UnPtrSurChose:= POINTER(ORD4(UnPtr));
```

ou, maintenant que le détypage nous est familier:

```
UnPtrSurChose := PtrSurChose (UnPtr);
```

Les fonctions ORD, ORD4 et POINTER nous sont familières. On peut les utiliser pour convertir des données de longueur identique ou différente:

```
var
    UnLongint   :   longint;
    UnEntier    :   integer;
    UnPtr       :   Ptr;
.....................................

aPtr        :=  POINTER(UnEntier);
aPtr        :=  POINTER(UnLongint);
UnLongint   :=  ORD(UnPtr);
UnLongint   :=  ORD4(UnEntier);  { paqué dans les }
                                 { octets les moins significatifs }
UnLongint   :=  ORD(UnEntier); { paqué dans les }
                                 { octets les plus significatifs }
UnLongint   :=  ORD4(UnEntier);  { paqué dans les }
                                 { octets les moins significatifs }
UnEntier    :=  ORD(UnLongint);
{ sont considérés les 2 octets de poids le plus faible }
UnEntier    :=  ORD(UnPtr);
{ sont considérés les 2 octets de poids le plus faible }
```

Un autre type, ProcPtr permet de manipuler les procédures et les fonctions comme s'ils étaient des **données**. N'est-ce pas étrange de voir à quel point le LisaPascal est parfois si rigide, et si perméable, en d'autres circonstances?

Si MaProc est une procédure définie dans le corps d'un programme, et UnProcPtr est un pointeur de type ProcPtr, alors:

```
UnProcPtr:=@MaProc;
```

retournera un pointeur dont le contenu sera l'adresse du point d'entrée de MaProc'. On pourra éventuellement s'en servir

comme paramètre passé à une routine en assembleur.

ProcPtr ne pointe pas sur la routine, mais sur un SignedByte, i.e. tout point d'entrée d'une jump table (Une table de renvoi vers d'autres routines. Voir le Segment Loader).

Enfin, on ne peut utiliser @ avec des procédures, ou surtout des fonctions imbriquées dans d'autres routines. Les programmeurs en assembleur sont moins concernés par le détypage, mais tout autant par le fait qu'ils peuvent, eux aussi, déréférencer les handle.

NEWPTR, DISPOSPTR

La fonction **NewPtr** sert à allouer un bloc. On obtient la taille de celui-ci en utilisant la fonction SizeOf. NewPtr retourne un pointeur (a longint) sur ce bloc.

```
UnPtrSurChose :=NewPtr(SizeOf(Chose));
```

alloue de l'espace mémoire dans le heap pour une nouvelle variable de type "Chose", et le pointeur UnPtrSurChose pointera vers un objet de ce type.

Ici, la place allouée est **non-relogeable**, puisqu'on a affaire à des **pointeurs**, pas à des handles. On peut aussi définir une taille **fixe**, en définissant ainsi une portion de mémoire (non-relogeable):

```
UnPtrSurChose :=NewPtr($1000);
```

Quand vous aurez fini de travailler sur un tel bloc, vous lib´rerez la place allouée par un:

```
DisposPtr(UnPtrSurChose);
```

N'utilisez plus, alors, UnPtrSurChose! Il pointera sur n'importe quoi, mais plus sur le bloc...

Pour ne pas tout mélanger, il est nécessaire de préciser qu'on peut utiliser des variables référencées par un pointeur **à l'intérieur** d'un bloc existant.

NEWHANDLE, DISPOSHANDLE

Ils ont la même syntaxe que NewPtr et DisposPtr. En se servant des mêmes définitions:

UnHandleSurChose :=**NewHandle**(**SizeOf**(Chose));

alloue un bloc **relogeable** block d'une certaine taille. NewHandle retourne l'adresse du **mp**. Pour en terminer avec l'espace alloué:

DisposHandle(UnHandleSurChose);

suffit.

Convertissez (détypez) ce handle si vous voulez l'utiliser avec une structure d'un autre type. Si UnHandleSurString (choisissez des noms plus courts, ça aide) se trouve être une variable de type StringHandle, on peut utiliser **n'importe quel type** d'identifiant comme une fonction:

UnHandleSurString:=StringHandle(nimportequoi);

EMBOUTEILLAGE MEMOIRE

Un point de plus avant de lâcher les freins: les routines d'OSIntf, ToolIntf et QuickDraw sont contenues dans trois principales librairies Pascal. Vous les trouverez dans l' Appendice, en fin de ce livre.

Les routines du MM peuvent: initialiser et allouer des zones de heap, accéder à celles-ci, purger des blocs. Elles offrent aussi quelques utilitaires.

Ne désespérez pas si vous n'en comprenez pas immédiatement la quintessence. C'est en voyant dans les programmes comment on les utilise que vous verrez réellement leur portée.

ERREURS COMMUNES, SUGGESTIONS

La taille requise pour l'allocation, c'est la taille d'un objet, ou encore SizeOf(ObjectHandle^^), et non SizeOf(ObjectHandle^), qui serait égale à 4 (octets), taille d'un pointeur.

Quant à ce qui suit, ce n'est pas d'un intérêt extraordinaire pour l'instant, mais vous saurez où le trouver au cas où:

```
UnHandleSurChose := NewHandle(Sizeof(UnChose));
```

ou encore:

```
UnHandleSurChose:= POINTER(ORD4(NewHandle(Sizeof(UnChose))));
```

Il s'agit d'une allocation mémoire et d'un détypage simultané.

Si vous développez des logiciels, c'est une bonne idée de créer des zones séparées pour les dialogues et les fenêtres. Ou de placer ceux-ci dans la pile, en les décrivant comme variables.

Si vous créez plusieurs heap indépendants, faites bien attention à ne pas détruire les blocs déjà alloués.

Placez tous vos blocs non relogeables aussi près que possible du début de la mémoire. Servez-vous de routines telles ReservMem.

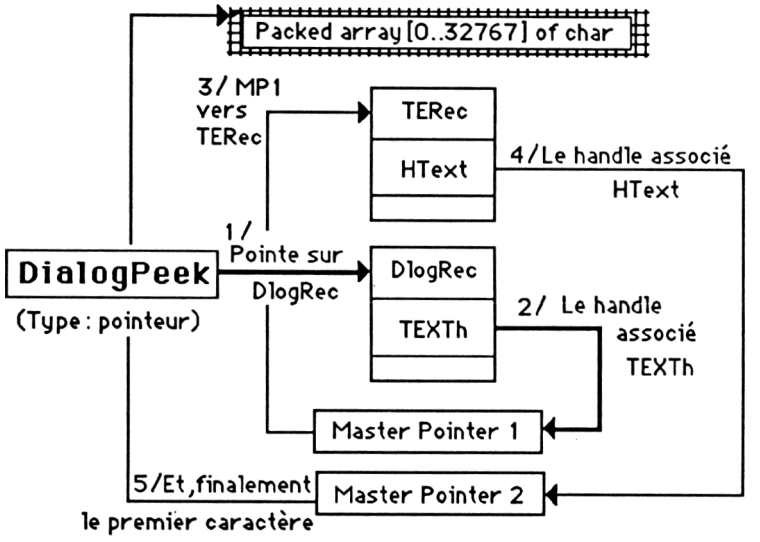
Au cas où votre application ne fait aucune acrobatie, ou de récursion (c'est très rare, même si vous croyez que non), définissez une taille de stack maximale, et ensuite, un SetApplLimit d'une zone maximale définie avec des ORD.

BOULOT DE SERRURIER: LOCK, UNLOCK

Vous devez **verrouiller** tout objet référencé par un handle, que l'on peut passer comme paramètre.

```
var
    MyHandle    :    Handle;
    Arrow       :    ^integer;
begin
    ....................
    MyHandle:=NewHandle(120); { réserve une portion de  }
    {          mémoire relogeable  de 120 octets de long }
    HLock(MyHandle);      { bloc verrouillé }
    Arrow:=POINTER(ORD4(MyHandle^));
    { dé-référencer avec le mp }
    Arrow^:=Arrow^+3; { ajoute 3 au premier octet du bloc }
    Arrow:= POINTER(ORD4(Arrow) + 5);
    { Arrow pointe à présent sur le 6ème octet du bloc }
    Arrow^:=Arrow^+3;     { ajoute 3 au 6ème octet du bloc }
    ....................
    HUnlock(MyHandle);  { libère le handle, le mp, }
                        { et le bloc }
```

Voici le chemin suivi par un pointeur sur une structure qui doit accéder à une structure imbriquée, et cela arrive couramment en utilisant TextEdit.



firstchar :=DialogPeek^.TEXTh^^.HText^^[0]

DialogPeek est un pointeur sur un DialogRec. TEXTh est un handle sur un TERec. HText est un handle vers un packed array of char. Suivez simplement le cheminement jusqu'au premier caractère du tableau... Et de toutes façons, il y a d'autres moyens d'avoir accès à un packed array of char - voir "TextEdit".

DES MACS...

Voici l'organisation de la mémoire du Mac, en commençant par la mémoire haute:

Buffer de son principal
Mémoire de l'écran
variables globales de l'application
Pile
Heap de l'application
Heap du système.

Ce n'est pas fini: dans le Mac + (1024Koctets de RAM), on trouve la table d'indirection du Toolbox, et d'autres variables globales.

Ensuite, on trouve dans les deux cas, les variables globales du Toolbox.

Puis, dans le Mac +, on trouve la table d'indirection de l'OS, alors que dans les autres modèles, on trouve la table d'indirection de l'**ensemble** ToolBox **et** OS.

Et enfin, dans tous les modèles, les variables globales de l'OS et les "trap vectors".

LE SEGMENT LOADER

Si vous êtes un vétéran de l'assembleur IBM ou du Cobol, les termes *overlays* et *segments* vous sont familiers. Cette technique, la segmentation, permet de faire fonctionner un programme alors que sa taille est supérieure à la taille de la mémoire. Ne pas confondre la taille programme sur la disquette, et la taille du programme: un petit programme de quelques lignes, qui ne ferait qu'allouer de la place pour d'immenses tables (arrays), n'occuperait que quelques Koctets sur la disquette. Mais pourrait fort bien dépasser la capacité de la mémoire.

En bref, segmenter un programme, c'est le tronçonner en

ensembles fonctionnels, qui sont appelés par le bloc d'action principal de programme. Certaines tâches ne seront appelées qu'une fois dans le programme (par exemple, une phase d'initialisation. Et des initialisations, il y en a dans un programme Mac. Et le segment de code sera **recouvert** par les segments qui suivent. Le but de la segmentation, c'est de:

- obtenir une programmation compacte, c.a.d. mettre beaucoup de code dans peu de mémoire (ancêtre de la mémoire virtuelle).

- modulariser les programmes: puisque chaque segment a une tâche bien spécifique, et que la boucle principale les appelle en séquence, un programme devient ainsi très facile à composer, à lire, et à maintenir (ou du mojns. le devrait!).

Sur d'autres machines, on avait l'habitude - ceux qui travaillaient en assembleur - de placer le segment *recouvrant* (overlapping) à l'adresse de début du segment *recouvert* (overlapped). Dans le Mac, il suffit de déclarer un segment purgeable. Lors du premier compactage de la mémoire, il disparaîtra. Quant au bloc principal, c'est le **Finder** qui le fait s'exécuter (quand, par exemple, vous double-cliquez sur son icône), en utilisant la routine **Launch**.

Les autres segments sont chargés au fur et à mesure qu'on en a besoin. Il suffit d'appeler une procédure ou une fonction faisant partie d'un segment pour que celui-ci soit appelé en mémoire (après avoir été lu sur le disque). Un segment peut atteindre la taille de 32 Koctets.

UnloadSeg

Une fois exécuté, le segment chargé sera rendu purgeable si vous appelez la routine **Unloadseg** du Segment Loader. Elle utilise un pointeur sur l'adresse d'une procédure ou d'une fonction.

ex: unloadseg(@Maproc);

GetAppParms

Cette procédure vous informera sur l'application en cours (nº de référence des ressources, etc.).

procedure

```
GetAppParms(var apName: str255; var apRefNum:integer;
            var apParam: Handle);INLINE $A9F5;
```

Elle n'est pas d'une importance majeure, aussi, nous l'éclipserons. Elle peut servir quand on imprime depuis le Finder.

Jump Table

Aucun problème si une routine est appelée par le segment qui la contient. Le Segment Loader est appelé de manière indirecte par **LoadSeg**. Et il se sert d'une **jump table** (table d'indirection), qui contient une adresse pour chacune des routines d'une application. Cette JUMP TABLE est créée par le **Linker**, qui construit un point d'entrée chaque fois qu'il rencontre un appel à une routine d'un autre segment.

Le Loader identifie les segments par le nº d'ID que le linker a attribué à chacun d'entre eux.

Chain

Le bloc d'action (d'appel) principal ne peut s'évacuer lui-même. Mais il peut appeler **Chain**, une routine qui lance une application depuis une autre, sans modifier le heap.

Pour mieux illustrer l'emploi d'UnloadSeg et vous plonger dans la programmation "à la Mac", voici un petit exemple de programme comprenant une phase d'initialisation, et une boucle d'événements:

```
program MonProg;

    const
    nbMenus = 3; { 3 menus, dont le menu Pomme }
    .............
    type
    .............
    var
    MesMenus: array[1..3] of MenuHandle;
    .............
```

```
        {$S Init}
        procedure  DrawMenus;
            ..............
            begin
        { insertion des menus, affichage de la barre de menus }
            end;

        procedure SetUp;
            ..............
        begin { SetUp }
        teinit;
        initdialogs(nil);
        initcursor;

        thesp := ord4(1024) * ord4(2048); { bon pour 2 Moctets! }
        freesp := compactmem(maxsize);
        freesp := maxmem(thesp);
        st_ptr := pointer($110); { début du stack }
        ali := st_ptr^ - ord(18772); { allouons ± 18 Ko de stack }
        setapplimit(ptr(ali));
        thesp := ord4(1024) * ord4(2048);
        hdl := newhandle(thesp); { un gros bloc! }
        disposhandle(hdl);
        FlushEvents(everyEvent,Ø);  { optionnel, mais il vaut }
                                    { mieux commencer avec }
                                    { une file d'événements vide }
            ..............
            end;{ SetUp }

        {$S }
{ ------------------------------------------------------------------------------ }

begin { MonProg, bloc d'action }
   moremasters;
   moremasters;
   moremasters;
   unloadseg(@moremasters);

   initgraf(@theport);
   initfonts;
   initwindows;
   initmenus;

   SetUp;
   UnloadSeg(@SetUp); { UnloadSeg évacue SetUp }
   freesp := maxmem(thesp);
            ..............
```

```
    repeat    { boucle d'événements }
    ..............
    until DoneFlag;
    ..............
end.{ MonProg }
```

Commentaires:

En tout premier lieu, on donne 3 x 64 mp aux programme. Puis, le programme initialise QuickDraw, et quelques autres managers, et ensuite, il exécute la procédure SetUp, qu'il évacue après l'avoir appelée. La procédure SetUp initialise quelques autres Managers, et gère l'allocation d'un immense bloc de mémoire.

Vous avez pu voir que le segment {$S Init} couvre plus d'une procédure. Il est préférable de grouper les routines dans des segments reprenant les mêmes types de tâches.

INTENTIONNELLEMENT LAISSE

EN BLANC

(POUR REMETTRE A BLANC

LA MEMOIRE...)

CHAPITRE V

Nous avons vu en détail la plupart des constructions, en Pascal, qui nous aideront à comprendre QuickDraw, et donc le Macintosh™.

Und **was ist** QuickDraw?

QuickDraw, c'est un ensemble prêt à porter de types de données, procédures, fonctions ayant trait au **graphisme**, à l'affichage sur l'écran. Affichage de n'importe quoi, d'ailleurs.

Ce qu'il est intéressant de noter, c'est qu'aujourd'hui, quasiment **tous** les ordinateurs sont graphiques (et non plus "text oriented").

QuickDraw affiche plus de 7000 caractères en une seconde. Ceci grâce aux quelques 145 routines, que vous pouvez appeler. Et qui sont testées.

VOTRE ATTENTION S.V.P.

QuickDraw n'utilise pas les variables Pascal *Ptr* et *Handle*, car il est indépendant du Heapzone Manager, où ces types sont définis. En revanche, il offre:

```
type    QDByte      = -128..127;
        QDPtr       = ^QDByte;
        QDHandle    = ^QDPtr;
```

QD préfixant les types indique qu'il s'agit de types "QuickDraw".

CONCEPÇION

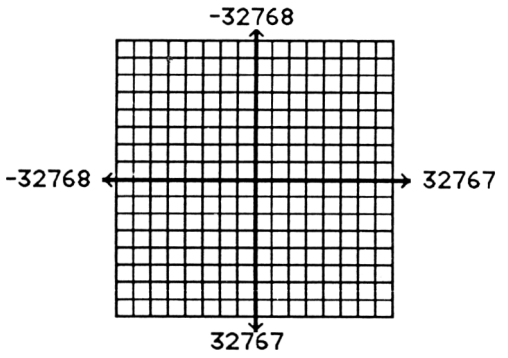
QuickDraw utilise quelques concepts mathématiques sur lesquels sont basées toutes les structures et les routines. Les voici:

- grille de coordonnées (-32768,..,0,..,32767),

- points infiniment petits (4,294,967,296).

GRILLE

Le point de coordonnées (0,0) se trouve au milieu de l'écran. Les directions croissent de la gauche vers la droite, et du haut vers le bas. **L'écran est en permanence une FENETRE sur cette grille**. Un peu comme le viseur d'un appareil photographique.



La grille de coordonnées

POINT

Chaque intersection de cette grille (à **deux** dimensions) définit un **point**. On peut le situer par ses coordonnées **h**orizontale et **v**erticale.

C'est un des types défini dans l'interface QuickDraw - dans un record dit "variant". Vous n'aurez donc pas à le définir vous-même.

Vous y avez accès sous deux formes: soit sous un ensemble de deux coordonnées (horizontale, verticale), soit sous forme de table:

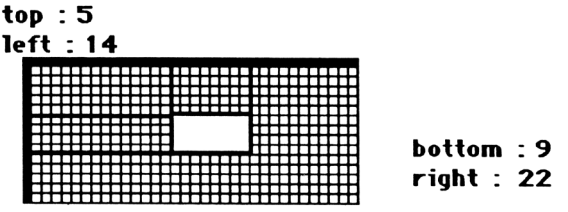
```
type
    VHSelect=(V,H);  { a selector }

    Point=record
            case integer of
                0:(v, h:integer);
                1:(vh:array[VHSelect] of integer)
          end;
```

RECTANGLE

Toute paire de points peut définir un **rectangle**, en se servant des positions **topleft** (haut-gauche) et **botright** (bas-droite).

Donc, le type **Rect**, défini dans QuickDraw, est constitué de deux **points** - ou quatre entiers:

Un Point, un Rectangle



On peut définir un point comme deux coordonnées de la grille.
Et un rectangle, comme deux points.

Le record variant suivant vous permet de considérer un Rect comme deux points, ou quatre entiers, ou deux tables de deux entiers chacune:

```
type

    Rect=record
            case integer of

                0:(Top:integer;
                   Left:integer;
                   Bottom:integer;
                   Right :integer);

                1:(TopLeft:Point;
                   BotRight:Point)
        end;
```

ou encore:

```
type
    Rect=record
```

```
        case integer of
            0:(Top,Left,Bottom,Right:integer);
            1:(TopLeft,BotRight:Point)
    end;
```

REGION

La **Region** est une structure spéciale. Le contenu d'une région est défini en plaçant **à l'intérieur** un point, une ligne, des figures (convexes ou concaves, en forme d'anneau ou disjointes).

Donc, un type **Region** n'a pas de taille définie a priori. Mais toute région peut être **entourée d'un Rect**(angle), une fois définie. Ce rectangle entourant une **Region** se nommera **rgnBBox** (region bounding box), dans la définition de la Region:

```
type
    Region= record
        rgnSize : integer; { 2 octets }
        rgnBBox: Rect;      { bounding box: un rectangle }
    { + une table de données de définition de la structure }
            end;
```

Dans l'exemple ci-dessus, on définit une region comme une structure ayant: une taille, un rectangle qui la circonscrit, et une table de définition des données.

On gère les Region de manière dynamique. On doit donc les accéder par pointeur. Quand une Region est déplacée, tous les pointeurs doivent être mis à jour. Il faudra donc se servir d'un handle spécial pour se servir d'une Region: un"RgnHandle".

```
type
    RgnPtr      = ^Region;
    RgnHandle   = ^RgnPtr;
```

rgnPtr pointe sur une Region et rgnHandle pointe sur le rgnPtr.

Vous rappelez-vous la "^"? Pour manipuler la rgnSize d'un **record** dont vous avez appelé le handle MonRgnHandle, vous

écrirez:

MonRgnHandle^^.rgnSize
Vous voyez, une fois de plus, qu'un handle est un double pointeur.

Nous venons d'évoquer les quatre concepts principaux de QuickDraw: **grille de coordonnées, point, rectangle et region**. Ce n'est pas eux que vous verrez sur l'écran, mais en utilisant de telles constructions, nous devons construire d'autres objets ayant une réalité graphique.

BITIMAGE

Un **bitimage**, c'est une collection de bits en mémoire que l'on peut stocker dans n'importe quelle variable statique ou dynamique, sous forme de **mots** arrangés en lignes de longueur (**row-width**). On peut organiser cette séquence de mots sous forme de *table* . La longueur d'une telle structure sera un **multiple** de sa largeur.

L'écran du Macintosh™ screen est un bitimage de 21888 octets correspondant aux 175,104 pixels (**pict**ure **el**ements). Un pixel est blanc si le bit correspondant est à zéro, et noir s'il est à un.

BITMAP

La définition du bitimage correspond à ce qu'on appelait le **bitmap**. Mais sur le Mac, un bitmap, ce n'est pas la collection de bits: un **bitMap**, c'est une structure dont la taille est **fixe**.
Par sa composition, il est relié au bitimage comme suit:

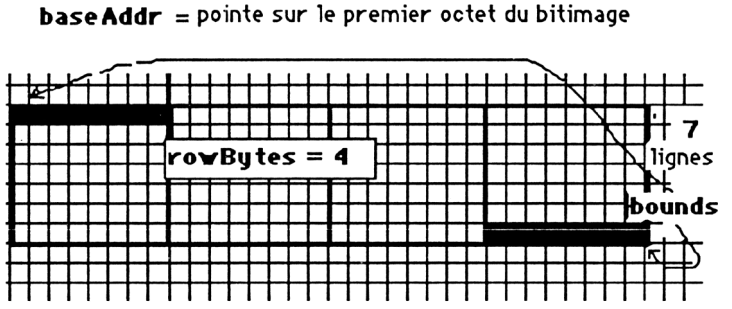
- un **pointer** vers le bitimage,
- la **row-width** du bitimage, en octets,
- et un rectangle qui l'entoure, défini par ses coordonnées.

C'est ce rectangle qui donne au bitmap ses **dimensions** et un système de **coordonnées**, de référence.

Une fois encore, le bitmap ne contient pas les points associés aux pixels. Mais il pointe sur la structure qui les contient:

```
type
  bitMap = record
           baseAddr    : QDPtr;    { un longint, 4 octets }
           rowBytes    : integer; { 2 octets }
           bounds      : Rect      { 8 octets }
         end;
                       { pour un total de 14 octets }
```

Le bitimage peut contenir du texte ou du dessin. La taille du bitimage n'a aucun effet sur celle du bitmap, qui est une structure **fixe**.



Un **BitMap** : **baseAddr, rowBytes, bounds**

Il **peut** y avoir plus d'**un** bitmap pointant sur le même bitimage, reliant celui-ci à diverses coordonnées.

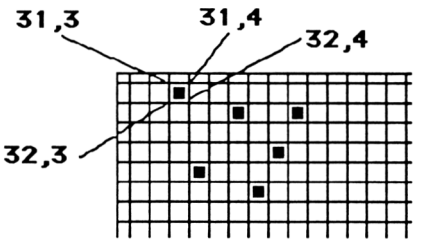
- baseAddr - pointe sur le début du bitimage en mémoire, et **rowBytes** est un entier qui contient le nombre d'octets de chaque **rangée**.

- bounds - c'est le rectangle qui circonscrit le bitimage. Le dernier bit du bitimage peut ne pas atteindre la frontière de mot. Le champ **bounds** coupera le dessin aux frontières du bit, dont les coordonnées sont stockées dans la définition du rectangle.

Il faut mentionner quelques particularités:

- les pixels qui correspondent aux points du bitimage sont situés **entre** les points du plan (grille) de coordonnées. La Fig.

ci-dessous illustrera fort bien le fait. Au fait, combien d'espaces y-a-t-il entre six allumettes? Cinq, naturellement. Vous retrouverez ces relations au moment où vous voudrez créer des dessins à l'écran...



Les points 'tombent' entre les lignes du bitimage...

- Un rectangle large de H points et haut de V circonscrit:

(H-1) * (V-1) bits.

L'angle (haut-gauche) du rectangle de limite entourant la region est externe au premier bit du bitimage. La largeur détermine le nombre de bits le bitmap possède dans une rangée.

Donc, la relation suivante est toujours vraie:

8 * map.rowbytes { nombre de bits par row-byte } n'est **pas plus petit que** **(map.bounds.right - map.bounds.left)**

La hauteur du rectangle détermine combien de rangées font partie du bitmap.

Pour que le nombre de bits du bitmap logique soit moindre ou égal à celui du bitimage, la relation:

sizeof(map.baseAddr^) **>=** (map.bounds.bottom - map.bounds.top) * map.rowBytes

doit toujours être vérifiée.

Sizeof(baseAddr^) est la taille du bitimage.

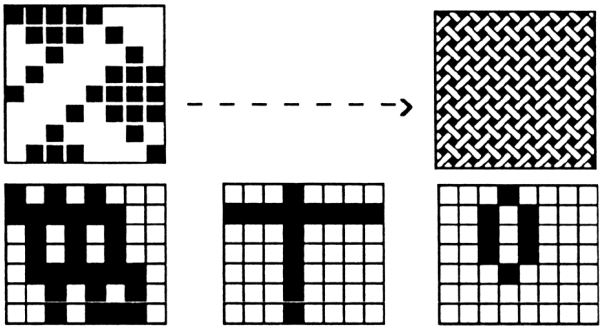
(maps.bound.bottom - map.bounds.top), et map.rowBytes, c'est la "taille" de chaque rowByte.

PATTERN (MOTIF)

QuickDraw peut remplir n'importe quel type de forme d'un motif. Celui-ci a une taille fixe (8 x 8 bits), et un rowWidth d'**1** octet (paddé à 2).

On utilise les motifs pour remplir des zones de l'écran, ou des dégradés. QuickDraw fournit quelques motifs pré-définis. Voici la définition du motif:

```
type

    pattern = packed array[0..7] of 0..255;
```



Exemples de Patterns (motifs)

CURSEUR, HOTSPOT

Est-il nécessaire de le définir? Vous le voyez si souvent sur l'écran du Mac, et savez que son mouvement est lié à celui de la souris.

Un curseur, c'est **aussi** un bitimage de 256 (16 x 16) bits, et de rowWidth 1 mot, ou 2 octets.

Lorsque vous amenez le curseur sur la barre de menus, vous voyez qu'il est entouré d'une fine ligne blanche. C'est la faute au **masque**, et si cette ligne qui entoure le curseur apparaît, c'est parce que le masque est plus large que le curseur (d'un bit, en ce

cas), sur toutes les faces.

L'apparence d'un curseur sur un écran est déterminée par l'interaction entre les bits du curseur, ceux de son masque, et ce qu'il y a "en-dessous" (c'est à dire ce qui se trouve sur l'écran. Quand les deux (masque et curseur) ont un bit commun (lisez, à la même position, à la même coordonnée), le pixel résultant apparaîtra **noir** sur l'écran.

Si le bit de masque est à zéro, l'apparence de l'image dépend alors du pixel correspondant sur l'écran, comme le montre le tableau suivant:

Données "dessous"	Masque	Pixel résultant
0	0	comme le pixel "sous" le curseur
0	1	Blanc
1	0	inverse du pixel "sous" le curseur
1	1	Noir

Voici la représentation hexadécimale/binaire d'un curseur en forme de X:

Hexa:	Binaire:
&H0000	0000000000000000
&H0000	0000000000000000
&H700E	0111000000001110
&H381C	0011100000011100
&H0E70	0000111001110000
&H0660	0000011001100000
&H03C0	0000001111000000
&H0660	0000011001100000
&H0E70	0000111001110000
&H381C	0011100000011100
&H700E	0111000000001110
&H0000	0000000000000000
&H0000	0000000000000000
&H0000	0000000000000000
&H0000	0000000000000000
&H0000	0000000000000000

Le code suivant donne l'équivalent du masque:

Hexa:	Binaire:
&H0000	0000000000000000
&HF00F	1111000000001111
&HE81F	1111100000011111
&H783E	0111110000111110
&H1FF8	0001111111111000
&H0FF0	0000111111110000
&H07E0	0000011111100000
&H0FF0	0000111111110000
&H1FF8	0001111111111000
&H783E	0111110000111110
&HE81F	1111100000011111
&HF00F	1111000000001111
&H0000	0000000000000000
&H0000	0000000000000000
&H0000	0000000000000000
&H0000	0000000000000000

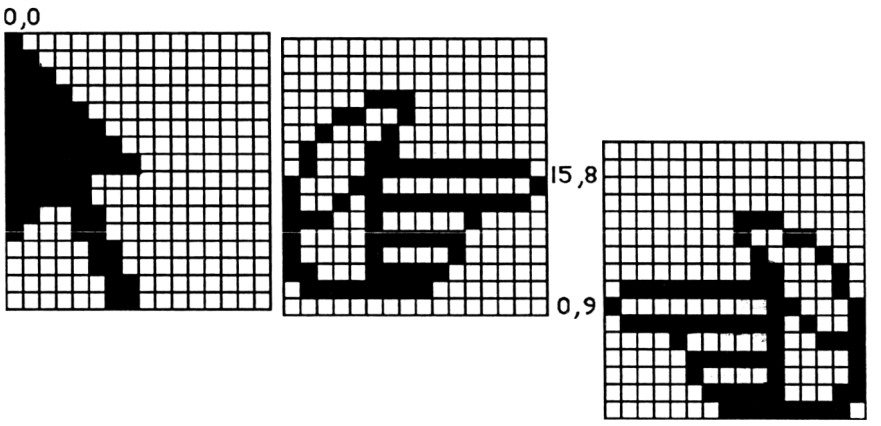
Si vous organisez les bits en groupes de 4, vous obtenez l'équivalent hexadécimal. Procéder ainsi vous fait aussi découvrir:

- un moyen pratique pour tracer des curseurs et des icones, et obtenir l'équivalent hexadécimal (utilisez quand-même l'icon editor" et affichez "Show Hex"!).

- intuitivement, le concept de "bitimage".

Afin d'aligner le curseur avec la position de la souris, il nous faut ajouter le **hotspot**. Celui-ci doit se trouver "sur" un objet (Menu, icone) lorsque vous cliquez dessus. Dans le curseur, il est défini comme un **Point**. Il a donc deux coordonnées, une horizontale, l'autre, verticale.

Les routines basses de la souris font constamment s'aligner le hotspot sur la position de la souris, et le curseur suit.



curseurs et hotspots

Dans la Fig. ci-dessus, les points référencés sont le "hotspot" du curseur.

Un entier (integer) contient 16 bits, et un hotspot est un Point, et voici notre type curseur défini:

```
type
    cursor=record
             data:    array[0..15] of integer;
             mask:    array[0..15] of integer;
             hotspot: Point
           end;
```

Le champ 'data', c'est le curseur. Chaque bit de chaque entier correspond au pixel que vous voyez dans la rangée. Il y a 16 éléments dans le tableau, chacun correspondant à une rangée.

LES 17 TRAVAUX DE QUICKDRAW;

Nous pouvons à présent faire un inventaire de ce que fait QuickDraw. On peut répartir les procédures et les fonctions en 17 catégories:

- action sur le curseur,
- dessin de lignes,
- affichage de texte,
- calcul de points,
- calcul de rectangles,
- action sur des rectangles,
- action sur des rectangles arrondis,
- action sur des ovales,
- dessin d'arcs,
- gestion de polygones,
- calculs sur des régions,
- dessin de régions,
- action sur des bitmaps,
- action sur des formes,
- interfaçage de routines,
- définitions et modifications de Grafports,
- utilitaires.

Munissons-nous de cordes et de pitons. Le pic est en haut, et nous forgeons nos outils. Celui que nous allons maintenant aborder est un des plus importants: c'est lui la base de tout environnement graphique.

CHOISIS TON CHEVAL:

GrafPort

"Graf" pour Graphiques, et "Port" pour port, on peut parier q'un GrafPort a tant soit peu affaire à la sortie de graphisme et traite de la manière dont s'effectueront les opérations graphiques.

Le GrafPort, c'est la structure (record) sur laquelle un programme bâtit les fenêtres, les dialogues, et plus généralement, les dessins. On peut définir plusieurs Grafports pointant sur le même bitimage, ouvrir plus d'un Grafport, sauter d'un Grafport à l'autre. On le nomme aussi *port*, tout court. A présent, énumérons ses caractéristiques, selon la définition de QuickDraw:

```
type
    GrafPtr  = ^GrafPort;        { Pointeur sur un GrafPort }

    GrafPort = record
        device   : integer;      { code du périphérique }
```

```
        portBits  : bitMap;       { bitmap du Grafport}
        portRect  : Rect;         { rectangle du port }
        visRgn    : RgnHandle;    { region visible }
        clipRgn   : RgnHandle;    { region de découpage }
        bkPat     : Pattern;      { motif de fond }
        fillPat   : Pattern;      { motif de remplissage }
        pnLoc     : Point;    { position du crayon graphique, }
                                  { en coordonnées locales }
        pnSize    : Point;        { taille du crayon }
        pnMode    : integer;      { mode de transf. du crayon }
        pnPat     : Pattern;      { motif }
        pnVis     : integer;      { "niveau de visibilité" }
        txFont    : integer;      { nº de ressource }
        txFace    : Style;        { style (sera expliqué) }
        txMode    : integer;      { mode de transfer / texte }
        txSize    : integer;      { taille du texte }
        spExtra   : longint;      { espace supplémentaire }
                                  { pour le texte proportionnel }
        fgColor   : longint;      { couleur du fond }
        bkColor   : longint;      { couleur premier plan }
        colrBit   : integer;      { "couche" de couleur }
        patStretch: integer;
        picSave   : QDHandle;   { picture being saved }
        rgnSave   : QDHandle;   { region sauvegardée }
        polySave  : QDHandle;   { polygone sauvegardé }
        grafProcs: QDProcsPtr;{ QDProcs; procs }
                                  { de bas niveau }
    end;
```

Ouf! Quelle liste! Que suggère-t-elle?

Promènons-nous parmi ces définitions sans notion préalable de ce qu'est un Grafport, et essayons de comprendre ce qu'elles signifient: nous connaissons déjà les types de référence, sauf **Style**, et **QDProcsPtr**. Ce dernier, nous le sentons bien, est un pointeur sur des **Procs**. Et qu'est-ce qu'une Proc?!

L'éthymologie de **fgColor** et **bkColor**, nous indique "foreground color" et "backGround color"; c'est réjouissant de savoir que la couleur est présente dans notre Mac, et qu'on peut la gérer.

En considérant les préfixes de chacun des éléments, on peut supposer que "pn" est le diminutif pour pen", "tx" pour "text", "visRgn" pour "visible Region" et "clipRgn" pour "clipping Region". Cet exercice nous renseigne déjà pas mal sur le GrafPort.

Entrons dans le détail:

- **device**

La sortie de graphisme ou de texte doit être dirigée sur un périphérique, écran ou imprimante, par exemple. Le numéro du périphérique logique de sortie est placé dans ce champ. Divers Managers le fixent à votre place, en utilisant les routines adéquates. En règle générale, il y a toujours une **routine** particulère pour placer une valeur dans un champ du Grafport.

Exemple: textfont(2), au lieu de MyDialog^.txface:= 2;

- **portBits**

Un bitmap pointant sur le bitimage courant qu'utilise le GrafPort. **portBits.bounds** est un Rectangle, qui entoure ledit bitimage. En modifiant ses coordonnées, vous changez le système de coordonnées du GrafPort. La taille par défaut est celle de l'écran du Macintosh™.

- **portRec**

Le **portRect** désigne la zone utilisée pour écrire du texte ou afficher des dessins à l'intérieur d'une fenêtre.

- **visRgn**

La **visible** Region indique quelle partie d'une region est visible sur l'écran. Un bitimage peut être plus large que l'écran (exemple: un dessin MacPaint), et sera réduit à sa visible region. Par défaut, la visRgn est initialisée au PortRect entier.

- **clipRgn**

C'est une region dont la taille par défaut est immense. Une application se servira de visRgn et clipRgn pour afficher dans un portrect. **visRgn** et **clipRgn** sont des regions, **pas** des rectangles. Ils peuvent donc avoir n'importe quelle forme.

- **BkPat, fillPat**

Certaines routines de QuickDraw utilisent **bkPat** et **fillPat**, deux champs de type **Pattern**.

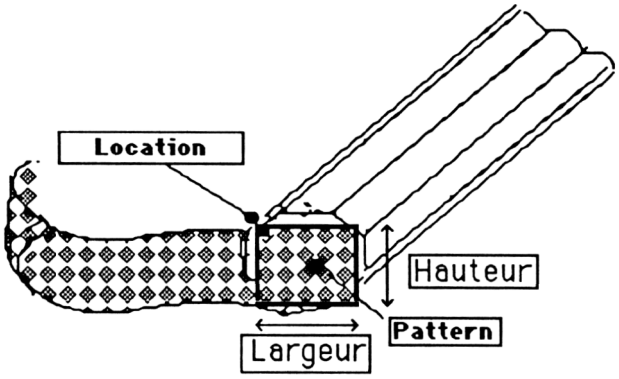
bkPat, motif de fond, est utilisé lorsqu'une zone d'écran a été effacée, ou déplacée.

fillPat, c'est la zone de stockage du motif utilisé pour remplir une forme **close**.

- **pen (size, mode, ...)**

Les cinq champs qui suivent, tous préfixés par "pn" ne signifient pas "pneumatique" ou "pneumonie".
Le GrafPort utilise un crayon rectangulaire **pen** (dont la longueur et la largeur peuvent aller de 0 à 32767 bits (donc, pixels), puisqu'il est défini comme un Point.

Sa taille par défaut est de 1 x 1 bit. Le coin topleft sert à positionner le crayon dans le système de coordonnées du GrafPort.



Un crayon graphique

- **pnMode** définit l'interaction entre le motif et les pixels présents sur l'écran, de la même manière que le curseur et le masque.

- **pnPat** détermines le motif de l'"encre". Un bon exemple, c'est la brosse dont vous vous vous servez dans MacPaint. Ce motif est lui-aussi aligné, par son coin **topleft**, au système de coordonnées du GrafPort.

- **pnVis** spécifie si l'"encre" est visible ou non.

Quant aux cinq champs à venir, ils spécifient l'apparence du texte dans une application.

Chaque caractère de texte est, en lui-même, un **bitimage** (jusqu'à 255). Chaque police de caractères contient un caractère par défaut, qui sera affiché chaque fois que le caractère frappé n'est pas présent dans la police choisie.

Au passage, avec une application comme le Font Editor (il faut savoir l'utiliser pour ne pas "bomber"), ou le Resource Editor, vous pouvez créer tous les jeux de caractères que vous désirez. L'en-tête de ce chapitre vous montre ce que l'on peut faire avec de telles "lettres".

- text (font, size, mode, ...)

txFont est un entier spécifiant le nº de police de caractère. Ø, c'est le nº de la police du système.

- **txFace** est défini au moyen d'un type "ensemble" (set). C'est le *Style* sous lequel apparaîtra le texte, **gras** ou *en italique*. Si vous vous servez de MacWrite, vous pouvez voir, dans le menu "Style", les possibilités offertes, ainsi que les combinaisons dérivant des types de base.

OBSESSION SETSUELLE

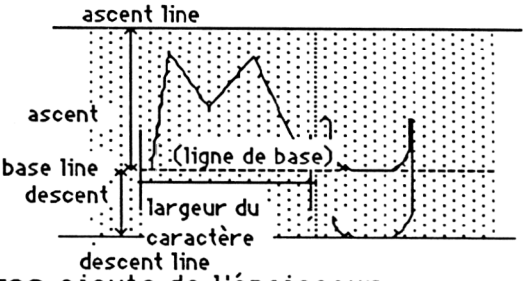
Le Style sera donc un "set" - ensemble - des styles offerts à l'utilisateur (ils sont rassemblés sous un seul type énuméré).

```
type
  StyleItem=(bold,italic,underline,outline,shadow,condense, extend);

    Style = set of StyleItem;
. . . . . .
```

Pourquoi un set? Vous savez qu'il vous est possible de **combiner** les styles de manière à obtenir de l'*italique souligné*,

ou du **gras** *ombré*. C'est la structure "set" qui le permet (l'ensemble vide étant l'écriture normale).

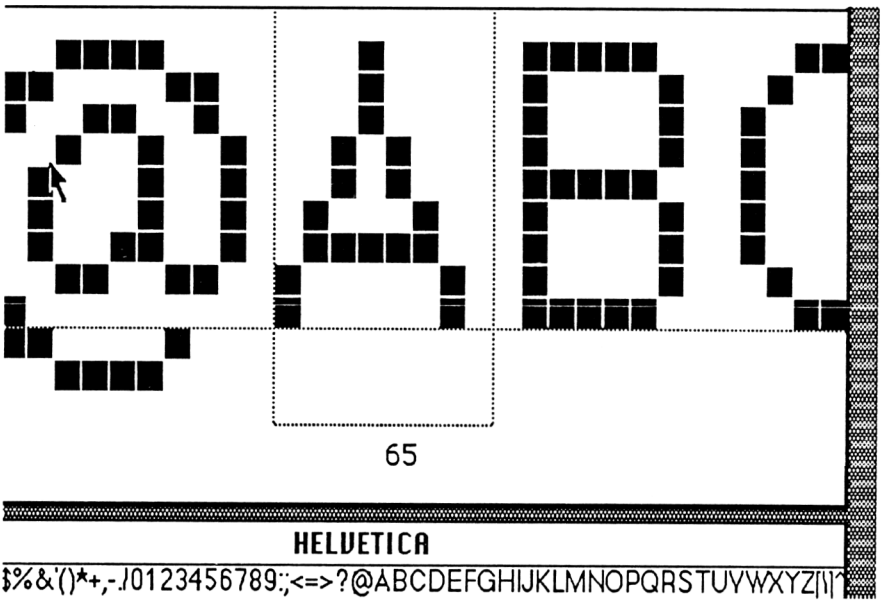
La figure qui suit montre les caractéristiques d'un caractère:



gras ajoute de l'épaisseur

En italique les bits des caractères situés en-dessous de la ligne de base sont décalés vers la gauche, ceux situés au-dessus de la ligne de base sont décalés vers la droite.

Ci-dessous, les caractères tel qu'ils appparaissent sous le FontEditor (utilitaire de transformation des polices de caractères).



- **txMode** définit l'interaction entre les caractères et les pixels déjà présents sur l'écran au moment de la frappe de caractère. Les **modes**: srcOr, srcXor et srcBic, analogues à ceux de pnMode, seront expliqués ultérieurement.

- **txSize** est un entier spécifiant, en incréments d'1/72ème de pouce, la taille des caractères. On peut choisir n'importe quelle taille. Même si elle n'existe pas, le Font Manager réévaluera l'échelle à partir du jeu de caractère dont il dispose dans une des tailles les plus appropriées.

- **spExtra** représente l'agrandissement du caractère "espace" nécessaire pour que l'on obtienne un espacement proportionnel. Lorsqu'est spécifiée l'option "Justifier", ou avec certains jeux de caractères, le caractère "espace" est augmenté d'un ou plusieurs pixels. Ce, afin d'aligner le nombre de caractères dont la largeur est inférieure ou égale à la largeur totale des caractères d'une ligne (calculé par le traitement de texte en additionnant, pour chaque ligne, la largeur de chaque caractère) avec la marge de droite. L'agrandissement dépend aussi des jeux de caractères utilisés (proportionnels ou non).

- **fgColor**, **bkColor** and **colrBit** stockent l'information nécessaire pour dessiner en couleur(s).

- **patStretch** est utilisé lors des impressions pour conserver à la sortie papier l'apparence obtenue sur l'écran.

- **picSave, rgnSave,** et **polySave** sont des handles. Lorsau'on travaille sur une region, on l'"ouvre", on décrit son contenu, on dessine en appelant les routines adéquates, et on l'affiche. Ensuite, on "ferme" la region (cad on **libère** la place occupée par la structure). Si une region est ouverte, **rgnSave** contient un Handle sur la structure, et autrement, ce champ contient **nil**.

Idem pour une "picture" et un "polygone": leur handle respectif sera stocké dans les champs picSave et polySave (nous les décrirons au chapitre VI).

- **grafProcs**, un QDprocsPtr, est une zone qui sert à "personnaliser" QuickDraw. **grafProcs** contient ordinairement **nil**, et QuickDraw fonctionne alors de manière standard. Nous ne nous étendrons guère sur les QDProcs, qui sortent du cadre de ce livre.

En observant ce qui se passe dans MacPaint, on comprend

bien QuickDraw: on voit par exemple des lignes brisées, une fois remplies d'un motif, faire une frontière parfaite: les motifs ne débordent pas (concept de region).

On voit alors que le crayon graphique peut avoir une taille variable (brosse, avec un motif...tout noir, en double-cliquant sur l'icone de la brosse).

On voit un motif s'appliquer au crayon graphique (brosse, avec un motif sélectionné).

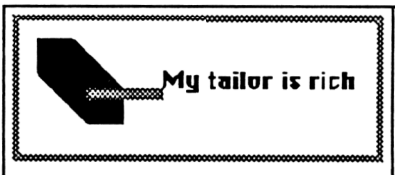
On voit la gomme, qui efface, et on comprend la superposition dee zones en mode *copy.*

Regardez les petites images que sont chaque caractère recouvrir d'autres zones.

Observez et découvrez par vous-même.

Et maintenant, une récréation avec le MacPascal. Il permet de tester les routines, le crayon graphique, et c'est bien agréable de voir un programme fonctionner, lorsqu'on l'a écrit soi-même.

Le programme qui suit dessine des lignes, utilise diverses tailles de crayon, affiche une ligne de texte et trace un cadre.



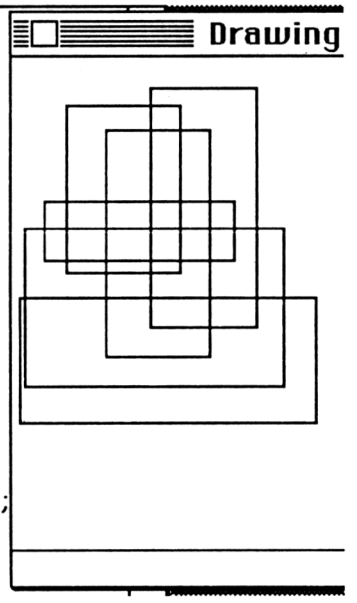

```
program Greg1;

  var
    MyRect : Rect;
begin
  pensize(15, 15);
  moveto(20, 20);
  lineto(40, 40);
  pensize(1, 5);
  penpat(gray);
  lineto(70, 40);
  textfont(0);
  drawstring('My tailor is rich');
  pensize(3, 3);
  MyRect.top := 10;
  MyRect.left := 10;
  MyRect.bottom := 70;
  MyRect.right := 160;
  FrameRect(MyRect);
end.
```

L'exemple suivant dessine des rectangles à partir de la position de la souris, lorsqu'on clique, jusqu'au point où l'on relâche le bouton. Puis, on dessine un rectangle noir. Observez le **mode**, changez pour celui que vous voulez, et voyez le résultat.

C'est facile, puisque vous constatez ce qui se passe immédiatement. Merveilleux d'avoir un Pascal interprété!

Notez le **repeat..until** imbriqué.

```
program RECTS;
  var
    P0, P1, P2 : Point;
    R : Rect;
begin
  repeat
    repeat
    until Button;
    GetMouse(P0.H, P0.V);
    P1 := P0;
    PenPat(Gray);
    PenMode(PatXor);
    repeat
      Pt2Rect(P0, P1, R);
      FrameRect(R);
      repeat
        GetMouse(P2.H, P2.V)
      until not EqualPt(P1, P2);
      FrameRect(R);
      P1 := P2
    until not Button;

    PenMode(PatCopy);
    FrameRect(R);
  until False
end.
```



A partir de cet exemple, vous pouvez fabriquer d'autres formes, avec FrameOval au lieu de FrameRect et dessiner toutes les formes que vous voulez sur l'écran.

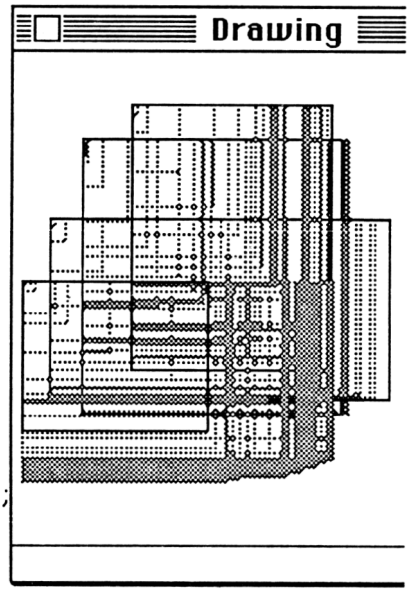
L'exemple précédent utilisait le mode PatXor. Voyez le résultat obtenu en changeant pour PatCopy, en page suivante.

Le "GetMouse" se sert de P0 (un Point) pour indiquer où se trouvait la souris au moment où le clic s'est produit.

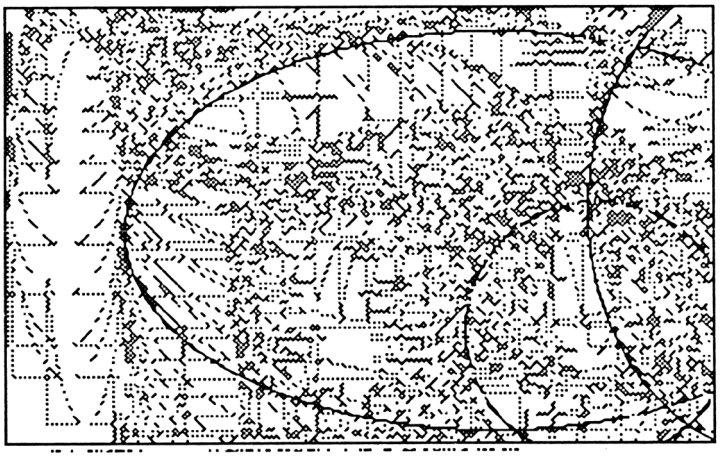
P0 est un Point. ses coordonnées horizontale et verticale seront, respectivement:

P0.h et P0.v

```
program RECTS;
  var
    P0, P1, P2 : Point;
    R : Rect;
begin
  repeat
    repeat
    until Button;
    GetMouse(P0.H, P0.V);
    P1 := P0;
    PenPat(Gray);
    PenMode(PatCopy);
    repeat
      Pt2Rect(P0, P1, R);
      FrameRect(R);
      repeat
        GetMouse(P2.H, P2.V)
      until not EqualPt(P1, P2);
      FrameRect(R);
      P1 := P2
    until not Button;

    PenMode(PatCopy);
    FrameRect(R);
  until False
end.
```

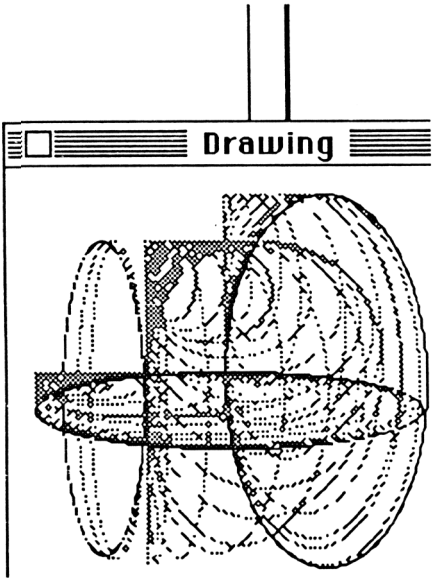


L'exemple suivant, lui aussi tiré du premier, utilise le FrameOval, mais en revanche, on garde le FrameRect intermédiaire. Voyez le résultat "psychédélique":



Ce dernier exemple ne garde que le FrameOval:

```
program Ovals;
  var
    P0, P1, P2 : Point;
    R : Rect;
begin
  repeat
    repeat
    until Button;
    GetMouse(P0.H, P0.V);
    P1 := P0;
    PenPat(Gray);
    PenMode(PatCopy);
    repeat
      Pt2Rect(P0, P1, R);
      FrameOval(R);
      repeat
        GetMouse(P2.H, P2.V)
      until not EqualPt(P1, P2);
      FrameOval(R);
      P1 := P2
    until not Button;

    PenPat(Black);
    PenMode(PatCopy);
    FrameOval(R);
  until False
end.
```



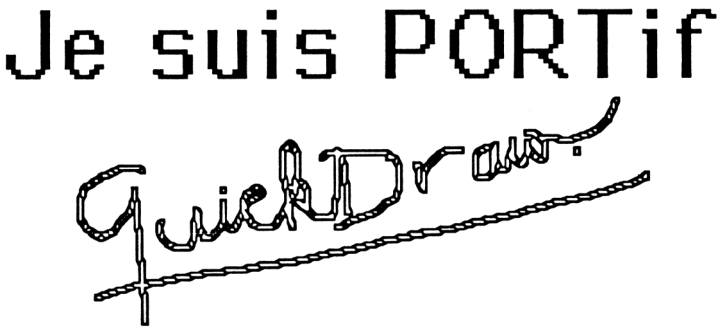
Jusqu'à présent, nous avons uniquement attaqué les notions théoriques.

Pour tant d'abnégation, vous méritez bien une récompense:

Je, Maître du Macintosh,
t'accorde une intelligence de 200,
et te nomme " L'astucieux."

CHAPITRE VI

PLUS SUR QUICKDRAW



Aujourd'hui, nous plongeons dans QuickDraw, dans ce qui en fait un outil unique, dans ses exploits.

Ensuite, puisque les fenêtres ont des liens étroits avec QuickDraw, et le port graphique, nous irons faire un tour du côté du Window Manager.

Révisons les notions couvertes au chapitre V, nous les utiliserons constamment par la suite:

DEFINITIONS

```
type
  QDByte   = -128..127;
  QDPtr    = ^QDByte;
  QDHandle= ^QDPtr;
```

```
VHSelect=(V,H);
Point=record
  case integer of
     0:(v, h:integer);
     1:(vh:array[VHSelect]
                of integer)
     end;
```

```
Rect=record
        case integer of
           0:(Top,Left,Bottom,Right:integer);
           1:(TopLeft,BotRight:Point)
     end;
```

Pour les amateurs de "C", les mêmes:

```
#define POINT union point
union point
   {
   short  v,h;
   short  vh[2];
   };

#define RECT union rect
union rect
   {
   integer rectpt[4];
   struct {
         short  top;
         short  left;
         short  bottom;
         short  right; } R0;
```

```
    struct {
        POINT topleft, botright; } R1;
    };
#define BITMAP struct bitmap
struct bitmap
    {
    long   baseAddr;
    short  rowBytes;
    RECT bounds;
    };
```

On peut considérer l'écran comme une suite de 342 lines de 512 bits (ou 64 octets, ou 32 mots).

Nous définirons déjà quelques routines dans ce chapitre. D'autres seront expliquées dans les lignes de programme. Et le reste sera mentionné dans l'Appendice B [QuickDraw]. Pour l'instant, nous complétons l'information fournie tout au long du chapitre V.

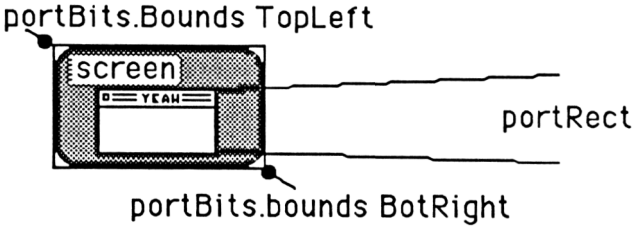
LUMIERES SUR LE PORT

Nous avons déjà utilisé les termes "coordonnées globales" et "coordonnées locales". La distinction est nécessaire, car chaque Grafport possède son propre jeu de coordonnées. Le champ portbits sert de système de référence pour un Grafport. Le rectangle portbits.bound, pour ne pas le citer, définit un ensemble de coordonnées locales: l'angle **topleft** est aligné sur le premier bit du bitimage sur lequel il pointe. Aligné, oui, mais à l'extérieur, puisque les pixels correspondants aux bits du bitimage sont **à l'intérieur de ces coordonnées**.

De fait, la base de tous les ports et des systèmes de référence, c'est le dessin lui-même, tranquillement assis dans son bitimage. Si on assigne les coordonnées (Ø,Ø) au premier bit du bitimage, nous créons des coordonnées **globales**. Une procédure, LocalToGlobal rend aisée cette conversion.

Les coordonnées du champ portRect sont dans le système défini par portBits.bound. C'est le lieu où tout est dessiné. Le dessin est limité par la visRgn, et apparaît àl'intersection de la clipRgn (types Rgnhandle).

Le dessin apparaît aussi là où se trouve le crayon graphique, et utilise les valeurs de taille, motif et mode du crayon (pnSize, etc.) contenues dans le GrafPort utilisé (actif).



Vous avez vu que clipRgn et visRgn sont de type *RgnHandle*, **pas** Rect. Donc, vous pouvez faire apparaître des frontières non rectangulaires (mais une region est toujours circonscrite par un rectangle).

Plus de rowWidth

Le champ rowBytes d'un bitmap est un integer (entier). Un rowWidth est un nombre de mots. Comme nous l'avons signalé, l'écran est un bitimage de342 lines de 512 bits, or 32 mots. La taille totale de l'écran (actuel - pas encore les 17 pouces attendus) est donc 10,944 mots.

L'écran du Macintosh™XL contient 364 lignes de 720 bits, ou 45 mots, pour un total de 16,380 mots.

Si un bitimage ne "remplit" pas la frontière de "droite" (c'est façon de parler, car en mémoire, cette notion de *droite* disparaît), ou si, afin d'avoir un nombre pair d'octets, un octet supplémentaire se trouve "paddé" (la rowWidth est comptée en octets), il contiendra des octets inutilisés.

Le champ **bounds** du bitmap, ce sera le rectangle dans lequel le bitimage sera circonscrit.

Certaines procédures QuickDraw permettent de fixer l'origine d'un port (voir ci-après dans le chapitre). Mais avant toute utilisation de QuickDraw, ses variables devront être initialisées.

C'est pourquoi on utilise **InitGraf** en début de programme.

InitGraf

QuickDraw est omniprésent dans le Mac; dans les dessins et le texte. C'est pourquoi on doit utiliser InitGraf. **Attention**, certains compilateurs "C" ajoutent du code et rendent inutile l'initialisation de certains Managers. Si vous utilisez un compilateur "C", sachez ce qu'il en est en regardant les exemples fournis sur vos disquettes.

InitGraf fait vivre thePort, initialise les variables globales et alloue de la mémoire. Voici sa syntaxe:

```
InitGraf(@thePort);
```

Le paramètre réclamé par InitGraf est un QDPtr qui indique à QuickDraw où il doit mettre les variables globales qu'il utilise. Parmi celles-ci, on trouve des nuances de gris, une "graine" (seed) de génération de nombres aléatoires, la variable screenBit, le curseur, et thePort, port graphique tous azimuths.

OpenPort, InitPort, ClosePort

Si vous voulez allouer de la place pour visRgn et clipRgn, et initialiser certains champs du GrafPort, utilisez "NewPtr" afin de crér un Ptr:

```
MonPtr      :=  NewPtr(sizeof(thePort));
MonGrafPtr  :=  GrafPtr(MonPtr);
OpenPort(MonGrafPtr);
```

(ou encore: MonGrafPtr := GrafPtr(NewPtr(**sizeof**(thePort)));)

InitPort s'utilise comme OpenPort, initialise tous les champs d'un Port, sauf clipRgn et visRgn. C'est la routine dont on se sert quand un Port a été désactivé. Si on appelle OpenPort, il n'est pas nécessaire d'utiliser InitPort pour le même port.

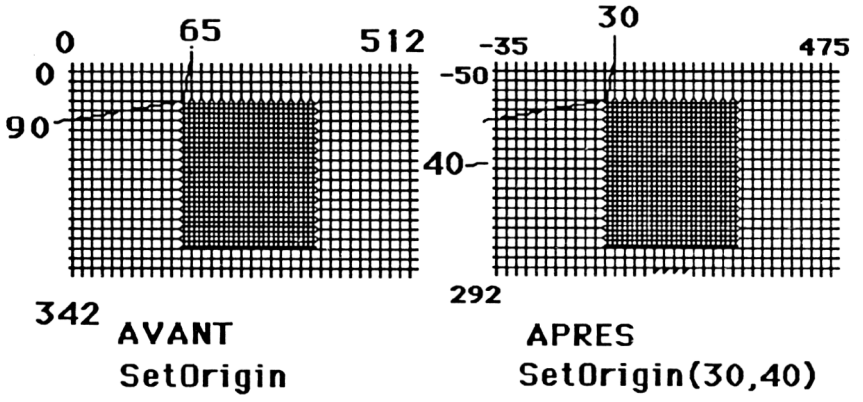
Quant à la routine ClosePort, elle libère la place occupée par les visRgn et clipRgn. Elle s'utilise comme OpenPort, mais à l'envers:

```
ClosePort(MonGrafPtr);
MonPtr:=Ptr(MonGrafPtr);
DisposPtr(MonPtr);
```

où MyPtr est de type Ptr.

SetOrigin, SetPort

Afin de re-fixer l'angle topleft du portrect d'un GrafPort, il y a SetOrigin. Le système de référence est alors modifiém et tous les points relatifs au portRect sont recalculés dans le nouveau système de référence:



Le dessin de gauche (ci-dessus) montre les coordonnées respectives (Ø,Ø) du rectangle de limite et du rectangle du port. Le dessin de droite montre les coordonnées recalculées après qu'on ait utilisé SetOrigin. Sur l'écran, rien n'a changé.

Avant d'utiliser SetOrigin, le port **actif** devra avoir été spécifié par la procédure SetPort, qui réclame un pointeur sur le GrafPort en question comme paramètre. Il est inutile d'utiliser SetPort après un OpenPort, ou un Initport.

Si MonPort est un GrafPtr, on peut écrire:

```
SetPort(MonPort);
SetOrigin(nouvTop,nouvLeft);
```

Il est important de redéfinir les coordonnées lorsque vous voulez utiliser un objet graphique appartenant à des ports différents, ou avant un scrollrect si vous avez une fenêtre immense.

Lorsque vous tracez une ligne, dans MacPaint, vous fixez la position du pnLoc (Point) au moment où vous **relâchez** le bouton de la souris.

Les entiers *pnMode* et *txMode* déterminent l'interaction des formes respectives dans le bitimage du port actif. La forme ou le

texte "amenés" à l'écran sont combinés, comme pour le curseur et son masque. On rencontre aussi cette notion d'interaction lors des transferts de bitimage.

Le *mode* de transfert, c'est le choix d'un parmi huit possibilités d'interactions, qui modifieront le bitimage récepteur lorsqu'il recevra des pixels supplémentaires.

Le bitimage sera modifié.

Modes de transfert

Si vous voulez transférer un bitimage entre deux bitmaps, ou écrire du texte, vous devez spécifier le mode de transfert. Le mode de transfert, c'est un moyen de combiner les bits d'un bitimage avec ceux déjà présents sur l'écran.

Copy, **Or**, **Xor** et **Bic**, voici les quatre opérations de base qui s'appliquent à chacun des deux types de transfert: txmode et pnmode.

Copy ne s'occupe pas du bitimage de *destination*. Il remplace les pixels du bitimage de destination par ceux du bitimage d'origine. Si, dans MacPaint, vous frappez du texte sur une zone déjà dessinée, le texte recouvrira l'arrière plan.

Dans le bitimage de destination, les modes **Or**, **Xor** et **Bic** ne modifient pas les pixels se trouvant "en-dessous" les pixels blancs du bitimage de départ. En d'autres termes, un pixel "blanc" d'origine laissera le pixel correspondant du bitimage de destination "tel quel".

Le mode de transfert "Or" n'ajoute un pixel noir que si le pixel du bitimage de destination était déjà noir.

Le mode de transfert "Xor" sert tout chaud un pixel "blanc" (bit correspondant du bitimage de destination à Ø) si les deux bits - celui du bitimage de départ, et celui de destination, étaient tous deux noirs.

Dans le mode de transfert "Bic", tout pixel noir du bitimage de départ devient blanc, et tout pixel blanc reste inchangé.
Chacun de ces modes de transfert peut être inversé par la négation *not.*, que l'on peut préfixer au mode de transfert: "patCopy", "notpatCopy", "srcXor", "notsrcXor", etc.

Couleur

Il est possible d'imposer une couleur de premier plan (fg) et d'arrière-plan (bg). Il est prévu jusqu'à 32 niveaux d'information de couleur par pixel. Mais à un moment donné, QuickDraw ne dessine que dans un seul de ces niveaux.

Pictures, Polygons

Tracer, du dessin ou du texte, c'est reprendre une série d'ordres de remplissage de bitimage, et d'affichage à l'écran. Si vous devez dessiner, un simple appel à toute une série d'ordres QuickDraw suffira. Cet appel, c'est une...PICTURE.

On accède une picture par un PicHandle, pointant sur un mp, PicPtr.

On peut définir les Polygons comme les Picture.

Des Polygones sont un ensemble de lignes fermées, vides, ou remplies d'un motif. Voici leur définition:

```
type
    Polygon=record
                polySize :    integer;
                polyBBox:     Rect;
                polyPoints: array[0..0] of Point;
            end;
```

On accède aux Polygons via un PolyHandle, pointant sur un master pointer PolyPtr.

StuffHex

En expliquant le curseur et le masque, au chapitre V, nous avons mentionné un moyen de représentation hexadécimale d'une forme. On peut utiliser cette méthode afin de stocker une séquence de bits dans un bitimage.
On pourrait placer dans MyImage:

```
MyImage[12] :=$BEAD; { les nombres en hexa sont }
                     { précédés  d'un '$' }
```

Lors de limpression, voici ce que cela donnerait: "• ••••• • • •• •".

Chaque chiffre hexa représente quatre bits, ou quatre pixels,

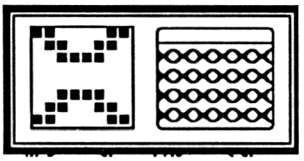
une fois affiché. Est-ce le meilleur moyen de dessiner?

Il existe une procédure, "StuffHex", qui vous permet de placer des bits directement dans un bitimage. Comme paramètres, elle demande un pointeur sur ce bitimage (afin que QuickDraw sache où agir), et une chaîne hexadecimale. Si on veut tracer une icone en forme de boîte, et si une icone est définie comme un array [0..95] of integer, on peut écrire:

```
Stuffhex(@icon  [0],'00000000000000000000000000000000000000000000000000');
Stuffhex(@icon[12],'0FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF0');
Stuffhex(@icon[24],'02000000000000000000000000000000000000000000000020');
Stuffhex(@icon[36],'02000000000000000000000000000000000000000000000020');
Stuffhex(@icon[48],'02000000000000000000000000000000000000000000000020');
Stuffhex(@icon[60],'02000000000000000000000000000000000000000000000020');
Stuffhex(@icon[72],'0FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF0');
Stuffhex(@icon[84],'00000000000000000000000000000000000000000000000000');
```

Si vous n'avez pas assez de place sur votre écran, prenez une paire de ciseaux, coupez la chaîne en utilisant l'instruction CONCAT, et fermez chacune des deux chaînes avec des apostrophes (quotes):

```
Stuffhex(@icon[0],CONCAT('0200',
                         ......
                         ......
                         '0000'));
```



Pour créeer le motif ci-dessus:

```
StuffHex(@thePattern,'C3663C00003C66C3');
```

Pour l'afficher, vous devriez écrire une procédure qui utiliserait **SetRect**, **OffsetRect**, et **CopyBits**. La voici:

```
procedure DrawIcon(h,v: integer);
{ DrawIcon dessine une icone }
var
  srcBits : BitMap;
  srcRect, dstRect: Rect;
```

```
begin
 srcBits.baseAddr:=@icon;
          { le champ baseAddr du bitmap (un QDPtr) }
          { pointe sur le bitmap de l'icone }
 srcBits.rowBytes:=6;
          { la largeur de l'icone est de 6 octets }

 SetRect(srcBits.bounds,0,0,48,32);
          { Cette procédure de QuickDraw }
          { fixe les limites aux dimensions }
          { de l'icone }

 srcRect:=srcBits.bounds;

 dstRect:=srcRect;
          { les rectangles source et destination auront }
          { les mêmes dimensions que srcBits. bound }

 OffsetRect(dstRect,h,v);
          { Cette procédure fait bouger le rectangle de}
          { h pixels horizontaux, et de }
          { v pixels verticaux }

CopyBits(srcBits,theWindow^.portBits,srcRect,dstRect,srcOr,Nil);
       { On dessine! theWindow^. portBits, c'est le portBit }
       { de la fenêtre (un bitmap) }
       { theWindow, ce n'est pas la fenêtre, }
       { mais un GrafPtr }
end;
```

Il existe bien d'autres moyens de dessinner. Qui n'utilisent pas StuffHex.

Au lieu d'utiliser setRect (troisième instruction dans la procédure), vous auriez pu assigner une valeur à chacun des composants de srcBits.bound:

```
with srcBits do
     begin
          bounds.left   :=  0;
          bounds.top    :=  0;
          bounds.right  :=  48;
          bounds.bottom:=  32;
     end;
```

setRect fait tout ceci en une seule ligne.

L'icone est lue depuis le fichier des ressources par le GetIcon.

PlotIcon dessinne l'icone. Ce n'était qu'un moyen de vous présenter StuffHex.

Appels divers

Si vous voulez tracer des lignes, vous vous servirez de Moveto, Line et LineTo.

MoveTo(h,v:**integer**); positionne le crayon graphique à l'endroit spécifié par h et v (en coordonnées locales). Il ne fait rien d'autre. On peut ensuite tracer des lignes, ou écrire du texte (Drawstring).

LineTo(h,v:**integer**) trace une ligne depuis la position du crayon jusqu'à l'endroit spécifié par h et v.

Vous voulez écrire "C'est moi le meilleur" sur votre écran, dans un **dialogue** de type *modal*, (il bloque tout jusqu'à ce qu'on lui fournisse une réponse), décrit dans le fichier des ressources, dont le no d'identification (voir Ressources) est 128?

```
var
   s: str255;
   ledial, saveport: dialogptr;
   litem: integer;
begin
getport(saveport);
ledial:=getnewdialog(128, ledialog, nil, pointer(-1));
setport(ledialog);
moveto(12, 15);
s:='C'est moi le meilleur';
DrawString(s);
repeat
modaldialog(litem, nil);
until litem = 1;
disposdialog(ledialog);
setport(saveport);
end;
```

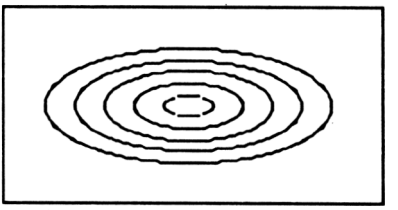
penMode fixe le mode de transfert. TextSize, la taille du texte.

Vous rappelez-vous le type énuméré avec lequel on définissait le "Style"? Le voici:

```
TextFace[italic];
MoveTo(156,170);
DrawString('Envoyez l''italique');
```

A présent, un peu de... Basic Microsoft Basic. On va s'amuser un peu avec QuickDraw. Ce programme trace des ellipses.

```
' Bienvenue aux planètes
CLS
FOR radius = 10 TO 60 STEP 12
   CIRCLE(100,60),radius,33,,,.4
   NEXT radius
' Frame, nicer
rectangle%(0)=20
rectangle%(1)=25
rectangle%(2)=100
rectangle%(3)=180
CALL FRAMERECT(VARPTR(rectangle%(0)))
END
```

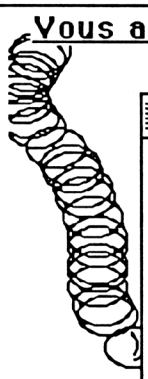


L'exemple suivant trace des motifs.

Il demande 4 valeurs, crée un motif de fond (backpat). Essayez avec des valeurs allant jusqu'à 32767.

```
' patterns
WHILE b <> 99
   GOSUB MyProc
WEND
END

MyProc:
CLS
FOR i = 1 TO 4
PRINT "item# "; i ;" = ";
INPUT  b
pattern%(i)=b
NEXT i
CALL BACKPAT(VARPTR(pattern%(0)))
RETURN
```

Dans le programme suivant, on teste le 390ème pixel pour afficher le texte:



Vous avez passé le 390è pixel

List

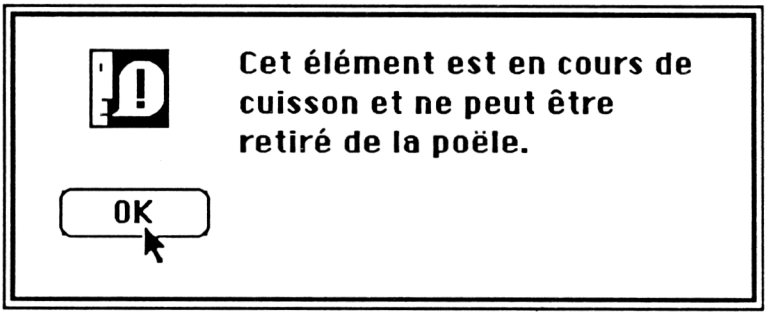
```
' dessinne un cercle de 12 pixels de
' large. Le centre de ce cercle se trouve
' là où vous avez cliqué.
CLS
a0:
a=MOUSE(0)
h=MOUSE(1)
v=MOUSE(2)
IF h<= 390 THEN aa
CALL MOVETO(10,10)
CALL TEXTFACE(5)

PRINT "Vous avez dépassé le 390è pixel"
aa:
IF a <> 0 THEN CIRCLE(h,v), 12
GOTO a0
```

Notez le *textface*, qui fixe l'affichage du texte en caractères **gras soulignés**. Avec QuickDraw, vous afficherez et dessinnerez ce que vous voulez, vous transporterez des régions, et comprendrez même comment fonctionne MacPaint.

Consacrez un peu de temps, faites des essais en "C", Modula II ou Pascal (LisaPascal, MacPascal, ou TML). Soyez curieux, amusez-vous!

Il y aurait beaucoup à dire sur QuickDraw, mais nous avons rendez-vous avec d'autres Managers.

L'AUBERGE MACINTOSH

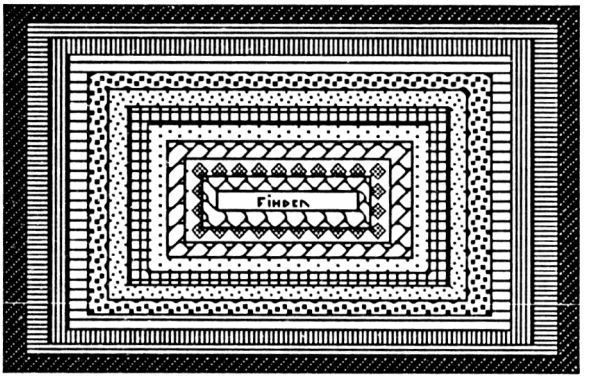


Prêt pour le combat contre le Finder?

CHAPITRE VII

LE FINDER,

LES RESSOURCES



Le Finder (trouveur!). L'avez-vous trouvé, au milieu du dessin? Non? achetez des verres de contact.

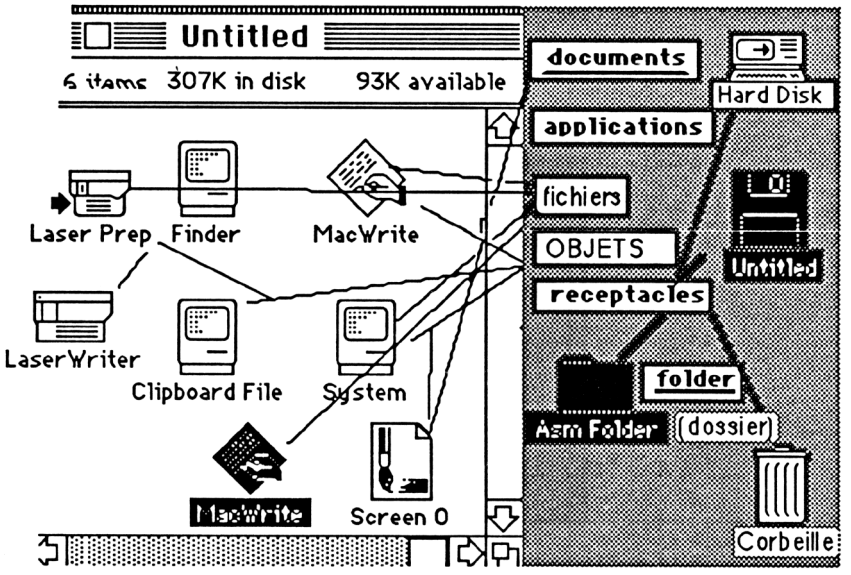
Nous avons déjà présenté le Finder comme étant l'**application** capable d'organiser, effacer, copier, transférer, renommer et afficher des fichiers.

Il vous permet de démarrer une application en double-cliquant sur son icone, ou sur celle d'un des fichiers (documents) qu'elle a créé.

Il fait tout cela, **sans aucun langage de contrôle**, dont avaient besoin d'autres systèmes d'exploitation, tel CP/M ou MS/DOS.

Toutes ces opérations se servent principalement de la souris, et quelquefois, d'un raccourci (Commande/Lettre), ainsi que du clavier - pour renommer un fichier.

OBJETS DU FINDER



Vous pouvez voir, sur la figure ci-dessus, quels sont les objets ordinairement rencontrés sur le DeskTop (bureau).

Les dossiers sont importants. Ils permettent de regrouper les données par **type**. Surtout sur disque dur, où le nombre de documents et dossiers peut être très important.

Les dossiers sont l'équivalent de la "racine" d'Unix.

SUR LE BUREAU

Que trouve-t-on sur le DeskTop (Bureau, dans un système français)?

- Tout d'abord, un **Menu**. Vous en connaissez les éléments constitutifs.

- Ensuite, les Accessoires de Bureau (Desk Accessories).

Le DeskTop contient aussi:

- des icones,
- une corbeille,
- un presse-papier (génial pour le passage de données intra-application, et d'une application à l'autre - mais on peut le court-circuiter pour passer des données),
- des fichiers, appelés aussi "documents".

LE FINDER, C'EST LA VIE

Le Macintosh™ ne peut être complètement dénué de **mode**. Parfois, le Finder, comme d'autres applications, affiche des boîtes de dialogue et d'alerte, lorsqu'il a besoin d'un complément d'informations. Celles-ci sont des "boîtes de dialogue" dites **modales**.

Fenêtres, icones, folders: caractéristiques

On sait ce que sont les fenêtres: ce sont des espaces qui permettent d'afficher du texte, ou des icones, dessins divers...

Vous avez certainement observé le fait qu'à un instant donné, **une seule** fenêtre est active, et que des fenêtres peuvent se chevaucher. Vous avez pu noter la succession de modifications qui s'effectuent, tant sur la fenêtre devenue active, que sur celles qui furent **désactivées** lorsque vous avez cliqué dans une zone de fenêtre inactive.

Une zone de fenêtre inclut son contenu, ainsi que les portions qui l'entourent et les contrôles.

On peut dupliquer des **dossiers entiers**. A l'unique condition qu'il reste sur le support (disquette, disque dur ou disque en mémoire - RamDisk) la place, au moins égale à celle du dossier. Ou que la directory (répertoire des fichiers) ne soit pas pleine.

Anticipez la position des éléments des boîtes de dialogues qui peuvent arriver sur l'écran: positionner le curseur en regard, ou presque de "OK" ou "ANNULER", vous fera gagner du temps.

Une application spéciale, chargée en mémoire en même temps qu'un application, c'est l'Accessoire de Bureau. Il exploite les possibilités "multitâches" du Macintosh™.

Nous avons vu les définitions des types des objets trouvés sur le DeskTop. Ces objets ont une forme, un nom, des *attributs*, un *emplacement* sur le bureau, et quelques autres données nécessaires aux fonctions du Finder. L'emplacement, vous l'avez remarqué, est **conservé** lorsque vous retirez une disquette du lecteur, puis la replacez. C'est donc que cet emplacement est gardé (sous forme de Point) dans une information fournie au Finder, qui le replace à l'endroit ou il se trouvait la dernière fois que le Finder avait noté.

Tout objet du Finder collectionne un certain nombre de données dans une structure, *Finfo*.

```
OsType = packed array[1..4] of char;

FInfo=record

        fdType:    OsType;      { type de fichier }
        fdCreator:OsType;       { créateur du fichier }
        fdFlags:integer;        { flags cad verrouillé, }
                                {est invisible, etc.}
        fdLocation: Point;      { emplacement du fichier }
                                { dans un dossier }
        fdFldr: integer;        {no de dossier contenant }
                                { le fichier }
    end; { FInfo }
```

RESSOURCES

Les Menus sont des ressources. Les jeux de caractères aussi. Les icones de même. Les fenêtres, les contrôles, les dialogues, leurs éléments, les alertes, certains fichiers.

Un programme Mac, c'est un ensemble composé de code et de paramètres, de jeux de caractères (qui sont, nous le savons, des bitimages, jusqu'à 255 caractères et/ou symboles par jeu de caractères). On y trouve aussi parfois des icones (32 x 32 bit), des masques, des curseurs, certaines d'entre elles, purgeables, d'autres, non. Et des définitions de fenêtres, de dialogues, de listes d'éléments de dialogues (DITL)...

On peut considérer que, pour le Mac, **tout** est ressource. En utilisant les ordres (routines) du Resource Manager (**RM**), on peut même créer ou modifier des ressources dans le fichier des ressources.

Une ressource de type **DITL** (Dialog Item List) ressemble à ceci:

```
Type DITL
* Définition de dialogues
     256,(32)
* DITL no 256, purgeable
* Les coordonnées de ce dialogue auront été définies
* dans le type DLOG correspondant;
     4
* 4 items - éléments
     BtnItem Enabled
* bouton actif
     420 275 500 295
* coordonnées: haut gauche droit bas
SUITE
* texte du bouton

     BtnItem Enabled
* bouton actif
     320 275 400 295
SUITE

* zone d'entrée de texte
EditText Enabled
40 40 55 150

* Texte affiché
StatText Disabled
70 40 84 145
Hello, World!
```

On se base sur 8 pixels par caractère quand on définit "au nez" un EditText Enabled, ou StatText, dans un DITL. Mais le "Dialog Creator", merveilleux petit utilitaire, a bien du diviser par deux les temps de développement de dialogues: il permet de créer en quelques minutes une fenêtre "vivante" - lisez, comme elle apparaîtra dans l'application -, et vous fournit en prime les *dimensions* du dialogue, et le positionnement de chaque élément de ce dialogue.

La syntaxe que nous montrons ici, c'est celle du RMaker. Il faut suivre certaines **règles** de syntaxe. Mais les disquettes de langages comportent toutes un format de chaque type, et vous pouvez acquérir la connaissance de cette syntaxe... en reprenant et modifiant les exemples fournis.

ATTENTION! Tout serait simple s'il n'y avait qu'un type de RMaker. Hélas, il y en a deux. Le premier, c'est celui du RMaker du Workshop. L'utilitaire REdit permet de *décompiler* en fournissant un fichier des ressources dans ce format.

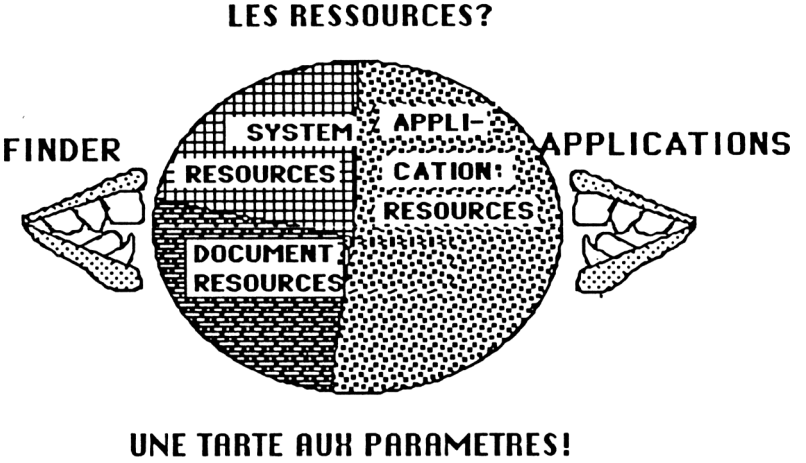
Le second, c'est celui du système de développement MDS que l'on trouve sur pratiquement TOUS les langages: Assembleur Consulair, Megamax C, TML Pascal...

Tout ce qui est défini dans ces ressources est un ensemble de paramètres utilisés dans le programme par les routines des différents Managers: le type DITL ici mentionné sera utilisé en programme par une routine du Dialog Manager:

```
MonDialog := GetNewDialog(256, NIL, POINTER(-1));
```

Parce que GetNewDialog ira chercher le type DLOG de no d'identification **256**, et le type DITL correspondant, l'affichera et sa place dans le heap (POINTER(-1)) sera conservée dans un dialogptr nommé MonDialog.

En plus de ses propres ressources, un programme peut utiliser les ressources du système. En général, il utilise les jeux de caractères du système, ou les ressources de type PACK (packages), ou INTL (international), pour les monnaies, les jours de la semaine et les mois de l'année.



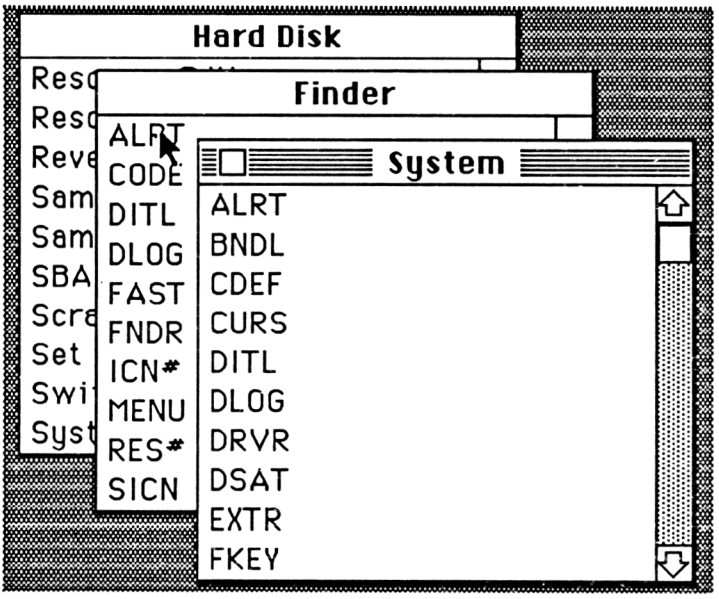
TYPES DE RESSOURCES / RMAKER

Le type de ressource s'écrit sur 4 caractères.

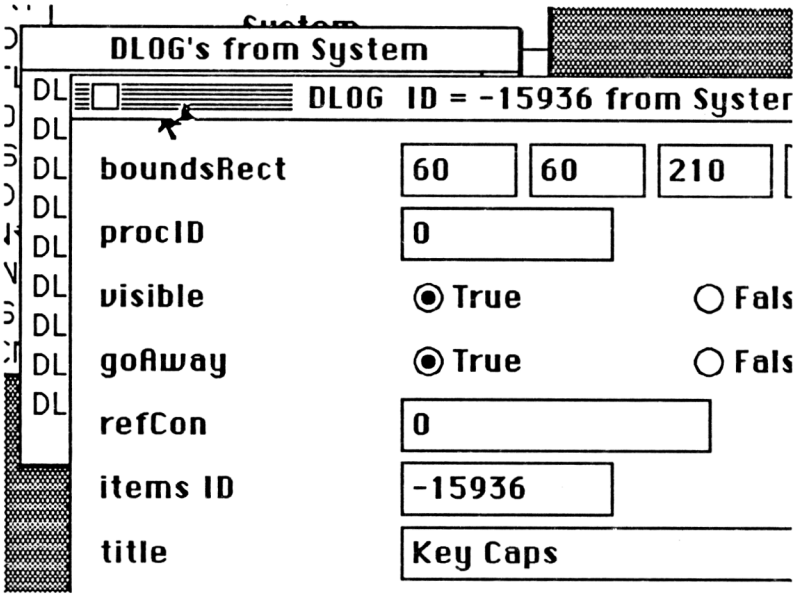
Dans la figure ci-dessus, nous avons mentionné les "ressources du document". Les documents, en général constitués de données, peuvent aussi comporter certaines ressources, comme un mot de passe, ou un jeu de caractères différent de celui du système.

Avec un utilitaire nommé "Resource Editor", ou ResEdit, vous pouvez vous promener dans les ressources des applications, documents et du système - Finder - DeskTop/Bureau de n'importe quelle disquette.

Sur la Fig. qui suit, voyez donc les ressources contenues dans le fichier système d'une disquette:



Ci-dessous, voyez ce qu'affiche le Ressource Editor, quand on lui fait avouer sans torture ce que contient un type DLOG.



Si les textes de vos logiciels ne sont pas assez parlant, vous pouvez les personnaliser. Toujours avec le Resource Editor.

Voyons maintenant la syntaxe des types les plus communéments utilisés:

Type de Ressource	Signification
ALRT	Une alerte
Type ALRT (32)	Toutes les alertes définies ci-dessous seront purgeables
,500	ID# 500
100 70 180 330	haut gauche bas droit
500	Associée à la liste d'items no 500 (Type DITL)
7761	chaque chiffre indique: *quand* utiliser le beep; le nombre de clics hors du bouton de l'alerte; le nombre de "bips".

ANYB Octets de paramètres. Peuvent être contenus dans un fichier.

Type ANYB	
AFile,289	Nom du fichier, ID#
32 512	Nombre d'octets - Déplacement depuis le début du fichier.

BNDL Information utilisée par le Finder.

Type BNDL	
,128	No d'ID
CDGG 0	Créateur (Propriétaire du Bundle)
2	nombre de types (cad APPL, WORD, INDX)
ICN# 1	Type, and nombre de ressources moins 1
0 128	No d'ID locale 0 'pointe' sur la ressource No128.
.....	définition d'icone et de masque.
FREF 1	File reference: Type et no de ressource.
0 128	Le No d'ID locale 0 pointe sur la ressource no 128. Vous pouvez placer plusieurs de ces définitions sur une seule ligne. En fin de chapitre, ces détails seront explicités. 'locale' signifie 'locale au FREF'.

C'est la connection entre un type de fichier et une icone.

CDEF définition de contrôles.

CNTL Liste de contrôle

```
Type   CNTL
    ,256                    No ID du contrôle
    ???                     Pas de titre
    -1 244 230 410          Dimensions du contrôle. Les
                            coordonnées sont locales à la
                            fenêtre
    Visible
    16                      C'est une barre de défilement.
    0                       Valeur du RefCon au départ.
                            Cette zone du record est traitée
                            par l'utilisateur.
    0  0  255              Peut prendre plusieurs valeurs, de
                            0 à 255.
```

CODE Segment de code de l'application.

```
    Type   CODE
        MyProg,0            Nom de la version linkée du
                            programme.
```

CURS Définition du curseur, valide pour **un** curseur. Rappelez vous, un curseur est une table de 16 x 16 bit. Le masque et le hotspot viennent ensuite:

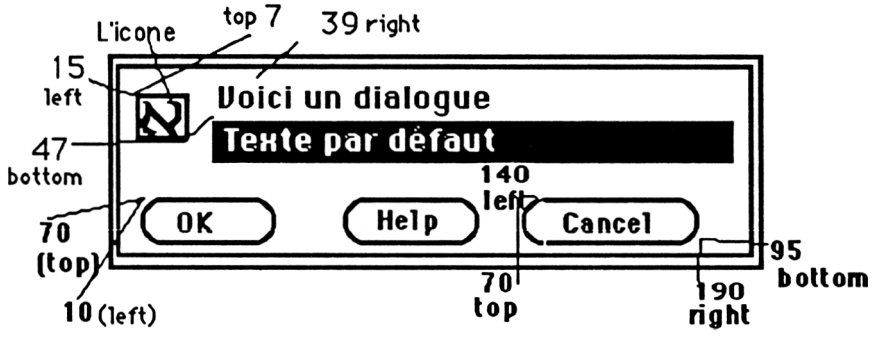
```
    Type   CURS
    ,256                    ID#.
00000000700E381CH0E70066003C006600E70381C700E000000000000000000000
0000F00FE81F783E1FF80FF007E00FF01FF8783EE81FF00F0000000000000000
*                           Curseur et masque
0007 0006           <----   hotspot, point actif du curseur.
```

DITL Liste d'éléments d'un dialogue ou d'une alerte.

```
Type   DITL
    ,500 (32)               No d'ID, suivie de l'attribut
                            'purgeable'.
    6                       6 items dans la liste dont le no
                            d'ID est 500

    BtnItem Enabled         Bouton actif
```

```
    70 10 95 50          Taille et position du bouton.
OK                       Texte du message.

    BtnItem Enabled
    70 80 95 120
Help

    BtnItem Enabled
    70 140 95 190
Cancel

    EditText             Texte pouvant être édité, comme
    4 50 18 300          par exemple "Save As..."
                         dans MacWrite.
    StatText Disabled
    24 50 38 250
    Voici un dialogue    Texte devant apparaître.

    IconItem Disabled    Une icone, dans le dialogue
    5 7 37 39            Notez les positions:

                         5 (Top); 7 (Left); 5+32 = 37
                         (Bottom); 7+32 = 39 (Right)
    256                  C'est l'icone no 256 qui sera
                         affichée.
```

Et voici ce qu'une telle définition afficherait:



DLOG Définition de dialogue.

```
    Type DLOG
* correspondrait au DITL précédent
*
    ,500 (32)
    50 10 100 310        Dimensions
    Visible 1 NoGoAway 0
    500                  pointe sur DITL no 500
```

sous le format RMaker du MDS, on devrait écrire:

```
Visible
NoGoaway
1
0
500
```

DRVR Accessoire de Bureau (Desk Accessory) ou I/O driver.

```
Type DRVR
     DeskAccFile!Test,286
```

FONT Définition de police de caractères

```
Type FONT
     TheFontFile!NewFont,64@0
     NomdeFichier!NomdePolice, no de ressource

     TheFont 10,64@10
```

Le no de ressource de ce type est bien curieux. Il est calculé de la manière suivante:

no = **(128 *** no**) + Taille du Jeu de Caractères.**

FREF File Reference, utilisé par le Finder.

```
Type FREF
     ,290
     APPL 0 Parms
```

FWID Largeur des polices de caractères.

GNRL Type général:

```
Type   GNRL
     ,430
     .I                  Entiers (définis en décimal).
3                        valeurs
25                       "    "
     .H                  Valeurs en hexadécimal
BEAFBØØBBEAD
AC1DDEAFDEAD
     .L                  Long mots (en décimal)
321422
     .S                  Chaîne de caractères
TheString
     .P                  Chaîne Pascal. Une par ligne.
```

```
.B                  Octets provenant d'un fichier.
Afont 64 256
```

HEXA Séquence de valeurs hexadécimales.

```
Type   HEXA
    ,377
CADD1E00
```

ICN#: liste d'icones

Type ICN# utilisé dans les BNDL et FREF, pour indiquer l'icone à associer à une application ou à un document.

```
,256
3                   3 icones seront définies ici.
0000 0000
0007 E000
.................   32 lines of 8 définitions hexadécimale
                    par icone.
```

ICON L'icone est définie comme une séquence de valeurs hexadécimales.

```
Type   ICON
```

* vous définissez votre icone. Servez vous de l'Icon Editor, et de son option Show Hex. Puis, retranscrivez les valeurs notées.

```
,256
0000 0000
0007 E000
.................   32 lignes en hexa.
```

INTL Ressources de type international, qui résident dans le fichier système.

KEYC Configuration du clavier.

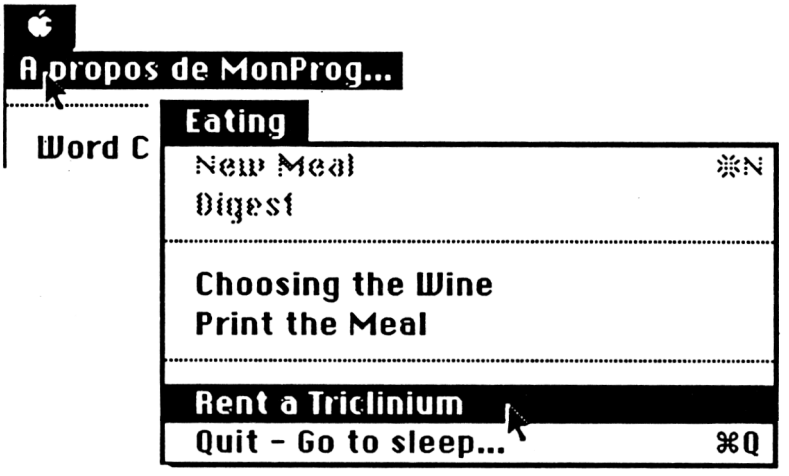
MDEF définition de la procédure de définition d'un Menu.

MENU Définition d'un Menu, et de son contenu.

```
Type   MENU
    ,1
    /14                     La Pomme.
    A propos de MonProg...
    (-
    ,2
    Eating                  Titre du Menu.
```

```
(New Meal/N     Elément, accompagné de
                l'abrégé Commande-N. La
                parenthèse gauche indique
                qu'il sera désactivé lors du
                chargement du programme.
                Par la suite, un ordre du
                Menu Manager (EnableItem)
                pourra le ré-activer.
(Digest
(-              ligne de pointillés.
Choosing the Wine
Print the Meal
(-
Rent a Triclinium
Quit - go to sleep.../Q
```

Voici le résultat:



PAT Pattern. Ordinairement suivie du Pattern List (liste de motifs)

```
Type   PAT
    ,266
    E70EF11FF00F00FF
```

PAT# Pattern list.

```
Type   PAT#
    ,266
    3                   3 patterns in this list.
    AA00AA11AA22AA33
    00AA11AA22AA33AA
    E70EF11FF00F00FF
```

PICT QuickDraw picture.

```
Type    PICT
    MyPict,256  (32)        Nom du fichier PICT, no d'ID,
                            purgeable.
```

STR Chaîne. Il s'agit en fait de STR suivi d'un espace...

```
Type    STR
    ,256
    Echantillon de texte...
```

On peut aussi écrire:

```
Type    STR
    ,256
3                           Number of strings to come
Chaîne no 1
Chaîne no 2
Ich been das string nummer drei
```

STR# Liste de chaînes de caractères.

PACK Package. routines résidant en RAM. Certaines d'entre elles ont regagné la ROM, sur Mac +.

WDEF Définition de fonction de fenêtre. Pas pour les débutants.

WIND Définition de fenêtre: no d'IDentification, titre, emplacement sur l'écran, ...

```
Type    WIND
    ,256
    La Fenêtre          Titre

    30 50 300 450       Dimensions (encore en top
                        left bottom right).

    Visible GoAway      La fenêtre est visible, et on
                        peut la fermer en cliquant
                        dans le carré en haut, à
                        gauche.
    0                   Type de fenêtre
    0                   procédure par défaut pour
                        gérer la fenêtre.
```

Attention:

- Les types énumérés ci-dessus sont des types de base. Vous

pouvez créer les vôtres, en les rendants égaux à l'un des types pré-définis. D'autre part, le RMaker du MDS ne permet pas certains types, CURS, par exemple. Vous créerez donc un tel type comme une ressource de Type CURS = GNRL, puis, la mention .H précédera les définitions du curseur, masque et hotspot en hexadécimal.

```
Type   CHSM  =   GNRL
   ,453
   .I              Entiers (en notation décimale).
   3                   valeur
```

- Les types suivants ont le même format:

CDEF, CODE, KEYC, MDEF, PACK, PICT et WDEF.

- PAT n'est pas le type de ressource, comme nous l'avons déjà dit,, mais PAT*b*, cad **PAT suivi d'un espace**.

LE RESOURCE MANAGER (RM)

Le RM conserve toutes ses informations dans le **fichier des ressources**. Le fichier des ressources du système est utilisé par toutes les applications.

Dans l'exemple utilisant GetNewDialog, vous avez pu comprendre comment un ordre du RM va chercher une ressource précise dans le fichier des ressources. Si le dialogue n'était pas décrit dans le fichier externe, votre programme se "gèlerait" au moment où vous utiliseriez GetNewDialog. Il vous est possible de créer un dialogue, même s'il n'est pas décrit dans le fichier des ressources.

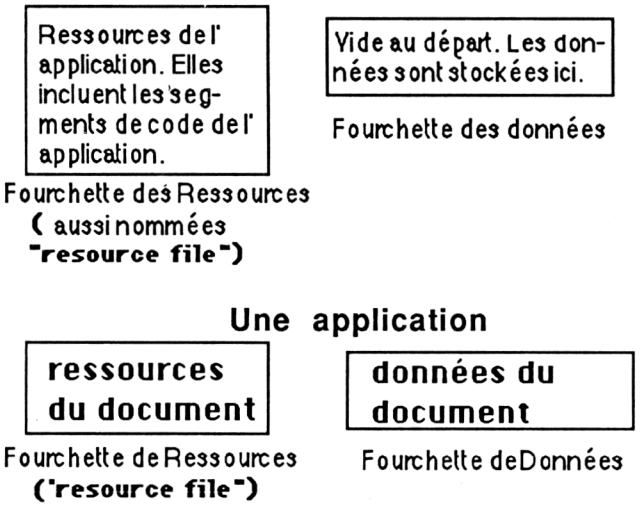
Vous vous servirez de la routine "**NewDialog**" et vous fournirez les paramètres nécessaires (comme par exemple la taille du dialogue, etc...). Seulement, si vous désirez changer quoi que ce soit, vous serez obligé de recompiler votre programme.

Ceci pour montrer encore une fois l'utilité des paramètres externes aux programmes, et non afin de montrer une routine du RM. Vous avez effectivement deviné que NewDialog fait partie du ??? Manager!

Les ressources sont donc lues - et mêmes écrites depuis - ou

vers - le fichier des ressources, via des routines spécifiques du RM. Le type et le numéro d'IDentification d'une ressource suffisent pour y accéder. Parfois, on peut aussi préciser un nom.

Lorsqu'on lui demande d'aller chercher une ressource, le RM la cherchera dans les ressources du document, de l'application, selon le cas. Et finalement, dans les ressources du système, s'il n'a rien trouvé.



Les ressources sont organisées sur leur fichier. Une série de pointeurs sait reconnaître son chemin parmi les divers types de ressources.

Un fichier contenant des données créées par programme verra celles-ci dans la **Data Fork**. Les ressources seront placées dans la "Resource Fork".

Lorsque, dans MacWrite, vous entendez tourner le lecteur de disquette, alors que vous frappez le premier caractère après avoir changé de police de caractère, c'est que:

- 1) le Font Manager a appelé le Memory Manager pour purger la police de caractères précédente.

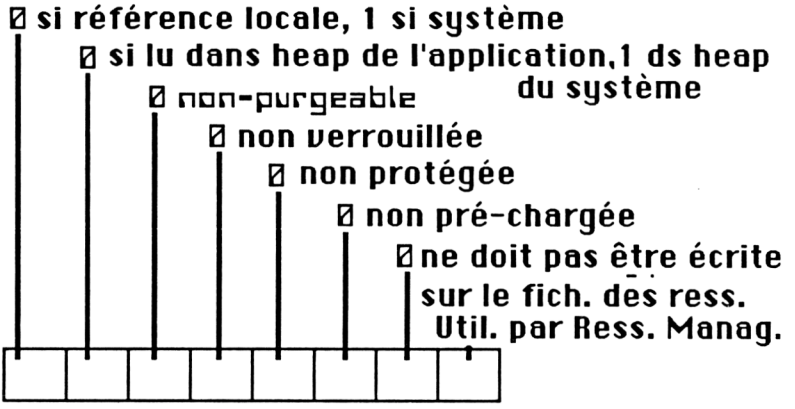
- 2) le Font Manager a appelé le RM pour charger la nouvelle police de caractères en mémoire.

Le fait qu'une ressource est verrouillée, purgée, reste ou non en mémoire, dépend d'une zone spécifique: l'**attribut de la ressource**. Le type de ressource est une séquence de 4

caractères:

```
type
    ResType = packed array[1..4] of char;
```

La Figure ci-dessous illustre l'attribut de chaque ressource. Une ressource, c'est, bien entendu, un ensemble TYPE / No d'ID.



L'octet attribut

C'est généralement le RM qui gère les modifications de l'attribut. Le bit pre-load (pré-chargée) indique au RM qu'il doit charger certaines ressources en mémoire au chargement du programme, plutôt qu'une par une.

ROUTINES DU RESOURCE MANAGER

On n'initialise pas le RM. Vous utiliserez **CurResFile** pour récupérer le no du fichier des ressources, **OpenResFile** pour ouvrir tout fichier spécifié par référence, et **CloseResFile** pour le fermer.

Pour accéder à une ressource dans un type donné, vous utiliserez **GetResource** ou **GetNamedResource.** Ces fonctions vous retourneront *theResource*, un handle sur la ressource.

Ces ressources, libérez-les d'un **ReleaseResource**. Cette routine fait appel à "DisposeHandle". Toute copie du handle (voir "déréférencer", au chapitre IV), sera invalidé.

La fonction **UseResFile** permet de modifier l'ordre de recherche des ressources: les ressources sont d'abord cherchées dans un document d'une application, puis, dans l'application elle-même, et enfin dans le système. Mais avec UseResFile, vous spécifiez dans lequel vous voulez que la recherche s'effectue.

Les ressources peuvent être modifiées en cours de programme. La modification d'une ressource, c'est soit le changement de textes, menus, etc..., soit un changement dans l'attribut d'une des ressources. Si vous voulez garder la trace de la ressource modifiée, des routines comme **SetResInfo**, **ChangedResource** et **SetResAttrs** s'en chargeront.

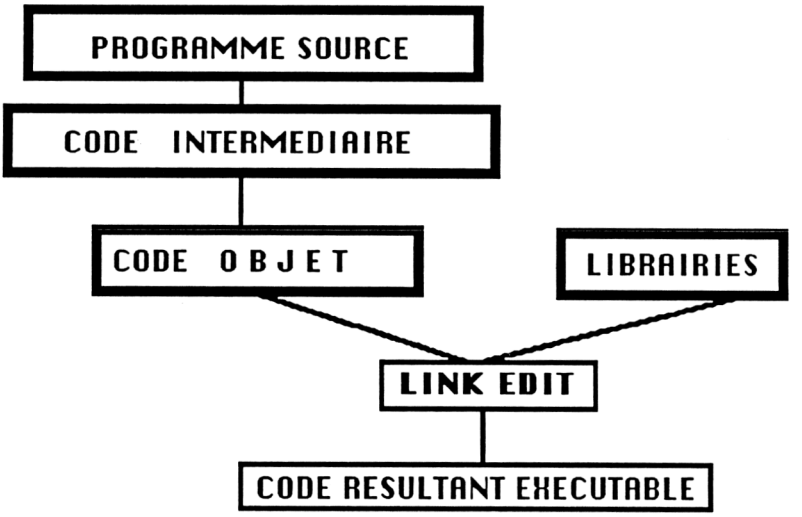
CountResources est une fonction qui rend un entier: le nombre de ressources du même type dans un ResFile (fichier des ressources) donné. Vous pouvez l'utiliser dans une boucle:

```
for i := 1 to CountResources do
    . . . . . .
```

Pour obtenir un handle sur une ressource, utilisez **LoadResource**(Handle). N'oubliez pas de verrouiller le handle avant de le déréférencer.

```
LoadResource(MyHandle);
Hlock(MyHandle);
MyPtr   :=   MyHandle^;
. . . . traiter la ressource . . . .
HUnlock(MyHandle);
```

VEUILLEZ SIGNEZ ICI "CREATEUR".

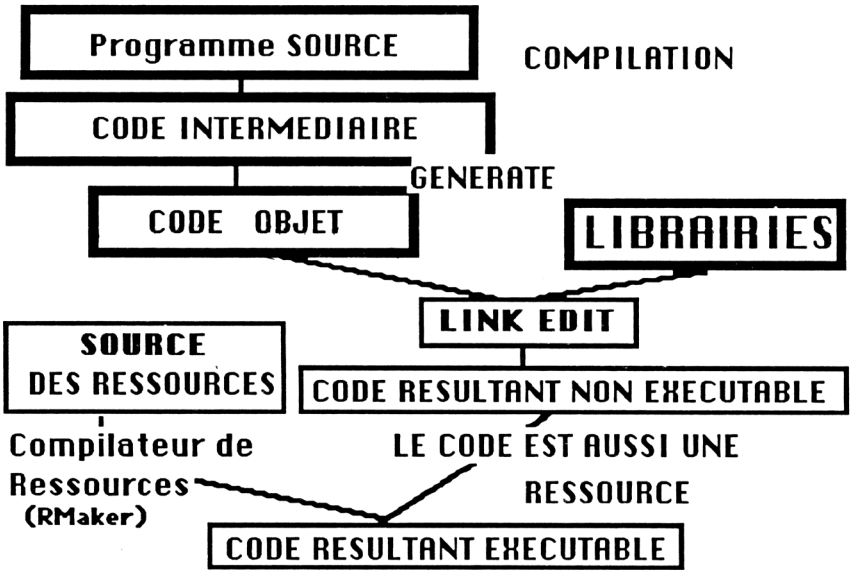


PROCEDURE CLASSIQUE DE COMPILATION

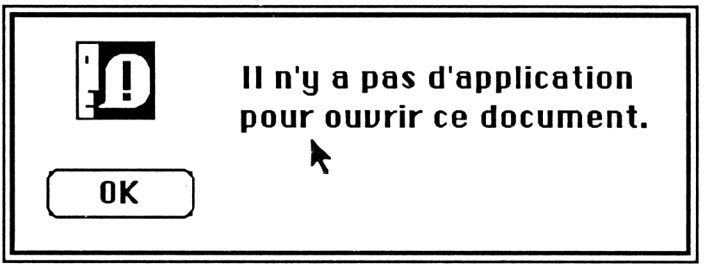
Le processus nécessaire à produire un programme Mac est commun à plusieurs langages: LisaPascal™, divers compilateurs "C", Modula II et l'Assembleur du MDS, en passant par le Pascal TML.

Ce processus est indiqué en page suivante. Sous Workshop, le code intermédiaire est traité par l'application "Generate" qui rend du code natif 68000.

En "C", et en Pascal TML, le code intermédiaire peut être de l'assembleur (une facilité supplémentaire pour apprendre l'assembleur), ou... rien du tout: on peut générer directement du code **.REL** (relogeable).



Vous savez que le Finder veut "savoir" ce qui relie une application et les documents qu'elle crée, ainsi que les icones qu'elle leur attribuera. Ce sont les spécifications du **type** et du **créateur** de l'application, ainsi que les ressources FREF et BNDL qui s'en chargent. Pour exécuter une application en double-cliquant sur une de ces icones, par exemple. Si l'application n'est pas présente, une alerte vous sera présentée:



Toute application est un créateur potentiel de documents. L'attribut "créateur", appartenant à une application, est unique à cette application. Apple gère les créatures - pardon, les "createurs", dans son service MTS (Macintosh™ Technical Support), et s'assure de l'unicité de chaque signature.

Un simple appel téléphonique, et vous êtes certain que les droits sur un programme ne vous seront pas disputés. Ou encore, qu'un double clic sur un document d'une application de type APPL, et de créateur "41424344" n'ouvrira pas une application qui, par hasard, aurait les mêmes types et créateurs.

"41424344" signifie ABCD en hexadécimal. Une signature comporte quatre octets, donc huit demi-octets. Un chiffre hexadécimale va de '00' à 'FF'. Cela fait pas mal de combinaisons possibles!

Quelques applications utilisent systématiquement la signature tou-venant "????". Vous remarquerez que l'on confond les termes "signature" et attribut "créateur".

Si vous utilisez SetFile, ou FEdit, ou encore ResEdit, vous pourrez parfois (pas toujours - dépendemment du format interne au document) réutiliser les documents créés dans une application par une autre application. C'est le cas d'Overview et Multiplan, etc...

VERSION DATA, FREF, BUNDLES

Dans le système, la signature est représentée dans la ressource de type BNDL. Dans le fichier ressource d'une application, il faut placer ce qu'on appelle les "données de version", **VERSION DATA**: c'est une chaîne de caractères qui indique le nom, le numéro de version, et la date de création. Cette ressource possède la signature de l'application comme propre type de ressource. Le Finder copie la Version Data dans le DeskTop (Bureau), un fichier invisible du système.

Quand à la forme des icones, elle sera contenue dans les ressources de type ICN# (voir res types de ressources ICN#, FREF et BNDL en pages 125 et 129).

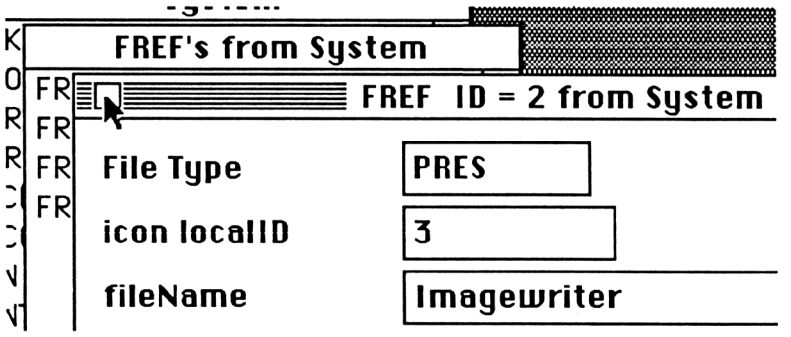
Si aucune forme n'est précisée, ni pour l'application, ni pour les documents qu'elle crée, les formes attribuées par le Finder seront les formes "standard" d'application et de document.

Application

Document

Lorsqu'une application est copiée sur un autre volume, le Finder doit savoir quels fichiers (s'il y en a) doivent aussi être

transférés.



Cette information, le Finder l'obtient d'une resource de type **FREF** (File Reference) se trouvant dans le fichier System. Chaque FREF contient un type de fichier et un numéro d'identification (locale à la FREF). C'est ce qui identifie l'icone à afficher. l'ID locale est reprise dans un type de ressource BNDL (bundle = paquet), qui contient:

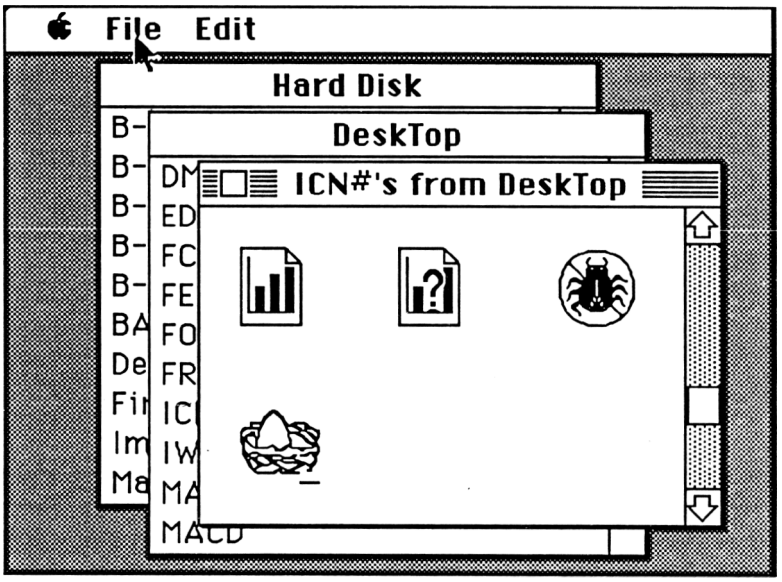
- La signature de l'application

- L'ID de la version data

- Une liaison entre les ID locales des listes d'icones (ICN#), tout comme dans les FREF, et les ID des listes d'icones des ressources de l'application.

- Les IDs locales des FREFs, ainsi qu'une liaison avec les ID des ressources (passage du local au global).

En utilisant le Resource Editor, ou ResEdit, fouillez dans les ressources du System, du Finder et du DeskTop, ainsi que dans celles d'une application qui servira d'exemple.

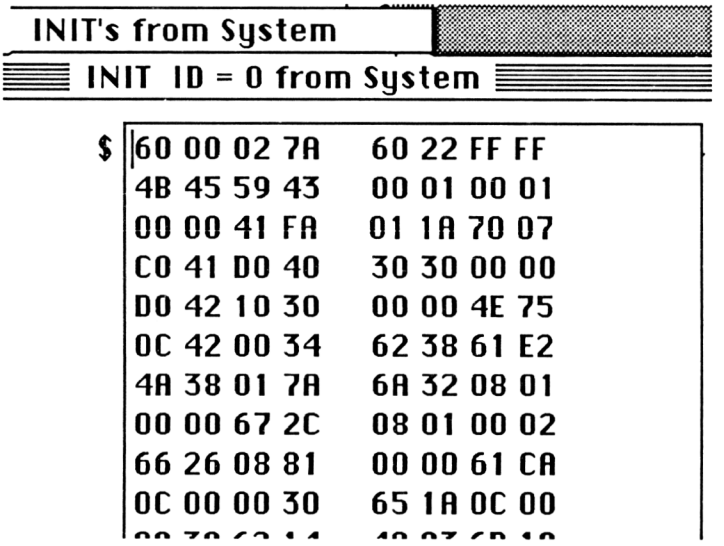


Lors de l'installation (ne serait-ce que par copie) d'une application, le bundle bit est mis à '1' et le Finder copie dans le Bureau les version data, liste d'icones, bundle, et FREF du fichier ressources de l'application.

Si, vous servant de ResEdit ou SetFile, vous modifiez le créateur, l'application perdra sa forme originelle, originale (sauf si l'application était dotée de l'icone standard). Tout lien entre les BNDL, FREF et l'application elle-même serait perdu. En remettant le créateur à ce qu'il était précédemment, vous retrouveriez l'icone primitive.

ANALYSE EN HEXADECIMAL

En considérant la masse de chiffres hexadécimaux qui suivent, vous vous apercevez que TOUTES les chaînes de caractères ne sont pas reprises dans des ressources de type STR ou STR#. Et quelques ressources peuvent être volontairement encryptées.



Si vous voulez comprendre le sens des chaînes hexa contenues dans la figure ci-dessus (cad A2A3A440617373) en les convertissant, voici deux programmes. Le premier, écrit en Basic Microsoft, version 2.0.

```
' convertit une chaîne de deux chiffres hexa en leur
' caractère ou symbole, ASCII ou non ASCII  *@#´éèçÎÏÊ
' ----*
lab0:
' ----*
   INPUT a$
   c$=""
   FOR i = 1 TO LEN(a$)/2
        b$=MID$(a$, 2*i - 1,2)
        b1$=LEFT$(b$,1)
        GOSUB perf
        b2=b1
        b1$=RIGHT$(b$,1)
        GOSUB perf
        b2=(b2*16) + b1
        c$=c$+CHR$(b2)
   NEXT
   PRINT c$
   GOTO lab0
' ----*
perf:
' ----*
   IF b1$ < = "9" THEN b1 = VAL(b1$) ELSE b1 = ASC(b1$) -
```

```
55
' Tout caractère frappé est censé être hexadécimal (contenu
entre Ø et F
    RETURN
```

Si vous avez le MacPascal, voici le même:

```
program ListIt;
   type
        SetOfNumChars = set of '0'..'9';
        SetOfHex = set of 'A'..'F'; (* c'est parlant *)
   var
        TheChar : char;
        TheString : packed array[1..80] of char;
        OddChar : boolean;    (* bool pour indiquer si premier ou second*)
                                    (* dans un groupe de deux *)
        i, j, k : integer;
        SetOfLetters : SetOfHex;
        SetOfDigits : SetOfNumChars;

 procedure readit;
 begin
   repeat
        read(TheChar);
   until ((TheChar in SetOfLetters) or (TheChar in SetOfDigits)
                                                    or (ord(TheChar) = 32));
   (* En MacPascal, 'Return' ou 'space' retournent une valeur de 32 *)

   if ord(TheChar) <> 32 then (* pas la fin de la séquence *)
        begin
             i := i + 1;
             OddChar := not OddChar;  (* inversion du flag *)
        end;
             if OddChar = true then
             k := 0;
(* k contiendra la valeur décimale de chaque paire convertie *)
   end; (* readit *)

begin (* main *)
(* initialisation des ensembles de référence pour validation d'entrée *)
   SetOfDigits := ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9'];
   SetOfLetters := ['A', 'B', 'C', 'D', 'E', 'F'];

 repeat (* main loop *)
   i := 0;
   OddChar := false;
   readit;
  while ord(TheChar) <> 32 do
```

```
    begin
        j := ord(TheChar) - 48; (* j contiendra des valeurs de 0 à 9 *)
        if TheChar in SetOfLetters then
            j := j - 7;              (* ou de 10 à 15 *)
        if OddChar = true then
            k := j * 16
              else                   (* i est pair *)
              begin
                  k := k + j;  (* k contiendra la valeur décimale des deux hex *)
                  j := i div 2;
                  TheString[j] := chr(k);
              end;
    readit;
    end; (* j tient la position du dernier caract. transformé *)
    for i := 1 to j do
        write(TheString[i]);
        writeln;
    until k = 255;    (* 'FF' pour finir, ou contiendra un ' *)
end.
```

Vous constatez qu'il est plus court d'écrire un programme en Basic. C'est parce que le Pascal est **structuré**, et comporte des délimiteurs de blocs, de programmes, de procédures. Aussi, il exige la définition des variables qu'il utilise.

Malgré tout, on peut écrire du Basic structuré. De toutes façons, le Basic existe, et c'est un bien: quelques têtes pensantes critiquent ce langage, prétextant qu'avec ses GOTO, il encourage les mauvaises habitudes mentales...

Beaucoup ont découvert la micro-informatique par le Basic, et ont ensuite changé pour le Pascal. L'environnement Pascal classique (compilation, link) aurait été trop difficile au départ.

Le Pascal compilé est beaucoup plus rapide que le Basic qui, lui, est interprêté. Le "C" est encore plus rapide.

Pour tester l'un de ces programmes, vous pourriez entrer la chîne hexadécimale suivante:

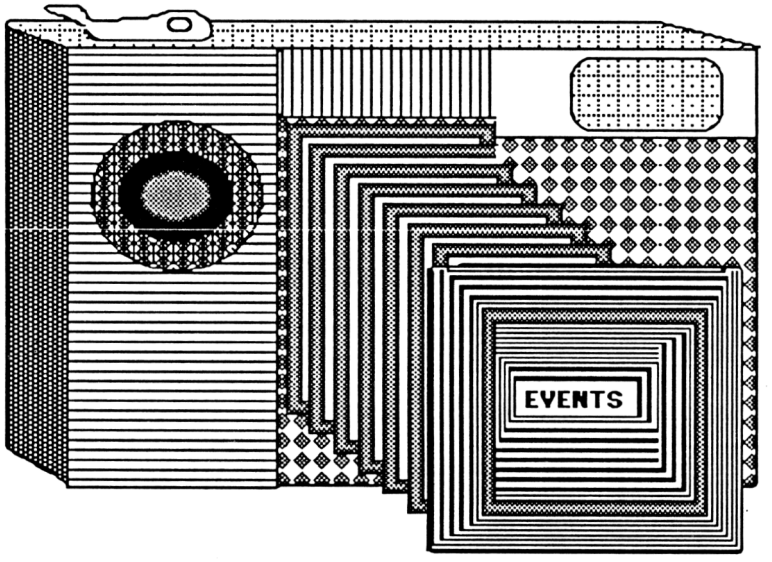
596F7520646964206974207420B8B7B5B4(CR)

A présent, nous allons enchaîner avec l'Event Manager, ensuite, nous rendrons viste aux gestionnaires les plus communément utilisés, et après TextEdit, nous **programmons**.

A partir de maintenant,
Tout vous sera FACILE
procedure Kw(ick,Dra:gon);

CHAPITRE VIII

FOCUS SUR LES EVENTS



Regardez ce qui se passe dans une application comme Word, qui permet à plusieurs fenêtres de s'afficher en même temps: malgré tout, une seule fenêtre est **active** à un instant donné. Il en est de même avec le Finder. Depuis cette fenêtre active, vous pouvez couper, coller copier, ou entrer du texte.

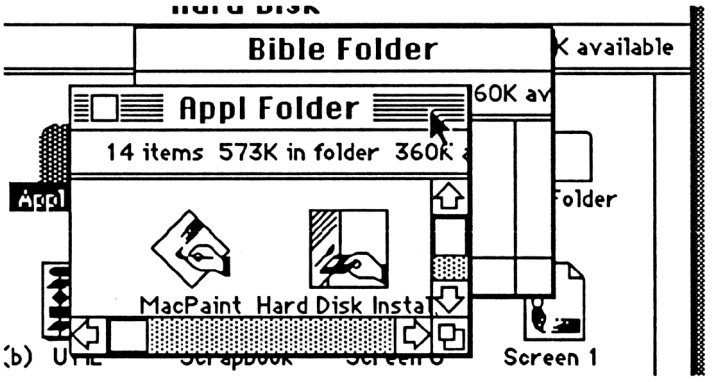
En cliquant dans la partie **visible** d'une autre fenêtre, vous l'activerez, tout en dé-activant la première.

Quelles transformations se sont en fait opérées?

En premier lieu, la portion active a été recouverte par la partie de la fenêtre maintenant active qui se trouvait "en-dessous".

La barre de titre de la fenêtre devenue active s'est ornée d'un genre de portée musicale - sans les notes de musique - et ses contrôles se sont remplis de grisé (s'ils étaient en grisé: cest le cas où du texte s'étend sur plus d'une page, dans MacWrite, ou encore, dans une fenêtre du Finder, si le nombre d'objets dans une fenêtre rend les barres de défilement - scroll bars - nécessaires).

Les contrôles, le titre, et la barre de défilement de la fenêtre dé-activée sont devenus blancs. La boîte de fermeture (en haut à gauche de la barre de titre), si la fenêtre en comportait, a disparu, ainsi que l'ascenseur.



Fenêtres se chevauchant

Créez, sous le Finder, quelques dossiers. Ouvrez-en deux. Faites une de leur fenêtre plus large que l'autre, rendez la plus petite active. Placez son angle haut-gauche sur celui de la fenêtre dé-activée, en la transportant.

A présent, cliquez dans la portion visible de la fenêtre non-active, et, immédiatement après, là où vous pensez que se trouve la boîte de fermeture.

Que se passe-t-il? La fenêtre activée a été rendue visible un court instant, puis, a disparu de la scène. Et cela signifie que:

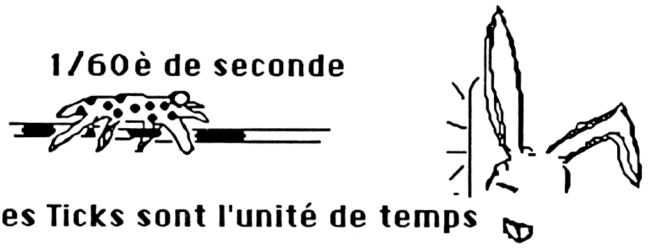
a) L'Event Manager a traité un événement de type mousedown sur la portion visible de la fenêtre inactive.

b) La fenêtre a été sélectionnée par un SelectWindow.

c) Le deuxième mousedown a été considéré comme étant dans la boîte de fermeture (GoAwayBox) de la fenêtre activée.

d) L'Event Manager a enregistré la position du clic de la souris (**where**) et après avoir mis à jour la fenêtre (Update), il l'a fermée. Ceci implique le fait que l'Event Manager (EM) savait que le clic dans la boîte de fermeture s'était produit **après** le clic sur la portion visible de la fenêtre inactive.

On peut en déduire qu'il garde trace de l'endroit où se produit tout événement, et du moment où il se produit.

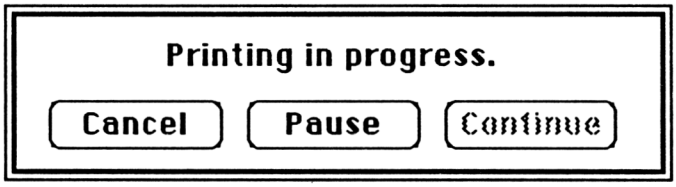


Ne les confondez pas avec les Bugs

Un utilisateur génère les événements. L'application tourne, et soudain, aux abris: l'utilisateur vient de bouger la souris, et a cliqué sur une fenêtre inactive; ou il vient de frapper une touche; ou encore, il a frappé une touche au clavier, inséré une disquette.

Tout programme Mac doit faire face à ces événements et prévoir de les traiter. Dans la mesure, bien entendu, oùi Is sont tous utiles: une application qui ne travaillerait que sur des fichiers, ne générant aucune fenêtre, ne traiterait pas certains types d'événements liés aux fenêtres.

La tâche de l'OS Event Manager est de renvoyer à chaque Manager chaque événement arrivé du hardware, et même, vers le ToolBox Event Manager. Le ToolBox Event Manager est précisément l'interface qui permet d'avoir notion des événements à l'intérieur d'un programme.
On peut aussi rencontrer des événements combinés: un "raccourci", un ensemble Commande-Majuscules-Lettre, etc...



En frappant "Commande-Point", on annulait une impression. Cliquer dans le bouton "Cancel" a le même effet.

Le type d'événement généré est appelé "abort-event", car il arrête le programme en cours. L'action à prendre sera choisie par le programmeur.

En maintenant une touche appuyée, vous générez un événement de type **auto-key**.

L'insertion d'une disquete provoque l'apparition d'un événement de type **diskinsert**.

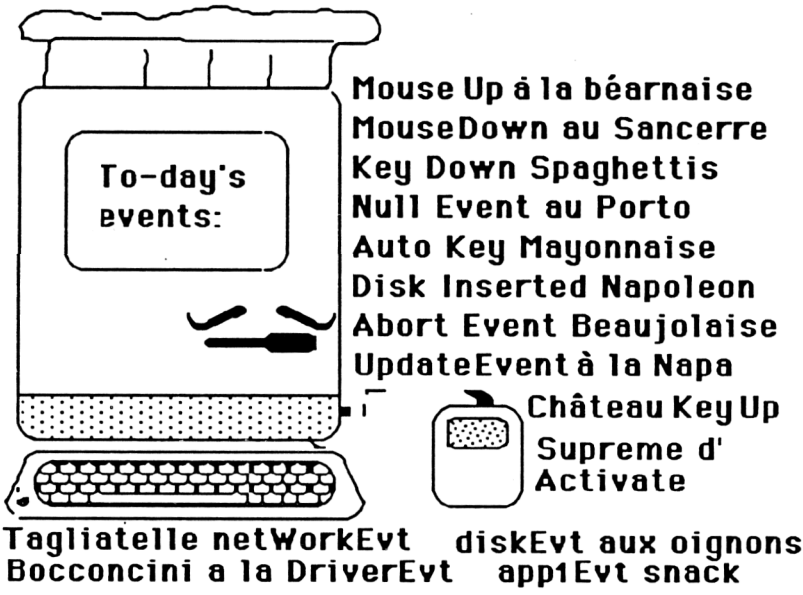
Lorsqu'on clique sur la souris, on provoque un **mousedown**. Lorsqu'on relâche le bouton de la souris, un **mouseUp**.

Dé-activez une fenêtre en activant une autre produit trois événements: un **deactivate/activate** simultané, et un **Update**.

Dans une application, un événement de type *activate* doit annuler toute manifestation visible d'une sélection dans la fenêtre qui devient inacive, et retracer les zones de texte ou de graphisme sélectionnées, dans la fenêtre qui devient active (comme par exemple, la sélection de texte, où la portion sélectionnée est en couleur inversée par rapport au reste du texte). Il faut mettre à jour les contrôles éventuels, etc...

Un événement de type *update* devra donner lieu à une mise à jour de texte ou de graphisme.

C'est le Window Manager qui les génère, mais l'EM les traite, par programmeur interposé: celui-ci doit prévoir les cas spécifiques à son application, et écrire les portions de code qui les gèreront.



Il existe deux types d'événements supplémentaires: **I/O driver** et **network.**

Enfin, comme quelqu'un qui dit qu'il n'a rien à dire, le **nullevent** est généré pour indiquer qu'il n'y a plus d'événements à gérer. Une routine d'ajustement de la forme d'un curseur peut être appelée au moment de la détection d'un **nullevent.**

Si le besoin s'en fait sentir, bien entendu. Un autre moment où une telle routine peut être lancée, c'est le passage d'une zone de fenêtre à une autre: dans une fenêtre de texte, la forme du curseur, c'est le I-beam (voyez dans MacWrite); en dehors, ce sera la fenêtre. Ces opérations ne sont pas automatiques, mais programmées dans une application.

L'EM peut enregistrer tous les événements qui se produisent, et les "rejouer". On peut se servir de cette caractéristique pour des démonstrations de logiciel, ou pour passer de l'information d'une partie d'une application à une autre.

PRIORITE D'EVENEMENTS, POSTING

Dès sa naissance, une fois reconnu par l'EM, un événement est placé dans une nursery appelée "file d'attente".

Le nombre de bébés admis ne dépasse pas trente, et ceux qui arrivent après chassent les premiers. Car la file d'attente est connue pour être FIFO.

On appelle **"timbrage"** (posting) le processus de passage d'événement à la file d'attente des événements. Certains sont passés au Desk Manager, et le programme n'en est pas conscient! Chaque bébé événement, comme toute jeune fourmi, a un rôle social à remplir.

Nous l'avons déjà dit, bien que la file d'attente soit de type FIFO, on peut traiter certains événements avant d'autres qui se seraient donc produits auparavant. On peut sélectionner un événement, ou plusieurs, grâce à un masque d'événements.

Des événements liés aux fenêtres peuvent être plus important que d'autres, en regard de l'application. C'est pourquoi une priorité a été assignée à chaque type d'événement.

La voici:

NIVEAU SUPERIEUR

- **activate,**

- **mouseDown**, **mouseUp**, **keyDown**, **keyUp**, **diskInserted**, **abort**, **netWork**, **I-O_Driver**, **application defined** (tous fidèles à la séquence FIFO),

- **autoKey,**

- **update** (ordre fenêtre de premier plan, puis d'arrière-plan)

- **nullevent.**

NIVEAU INFERIEUR

Les événements de type Activate/de-activate ont la priorité la plus haute: ce qui se passe dans un programme dépend de la fenêtre active.

Les événements de type *keyboard* contiennent aussi des informations comme le code ASCII généré par la touche frappée, et le code de la touche. Bien entendu, le code ASCII dépend aussi du clavier local: sur un clavier américain, le code généré en

frappant la lettre 'Q' sera 81. La même touche frappée sur le clavier français générera le code 65, correspondant à la lettre 'A'. Mais le code de la *touche* sera le même dans les deux cas.

Toutes les touches, sauf Majuscules, Verrouillage des Majuscules, et Commande (celles-ci sont appelées *modificatrices*), génèrent des événements lorsqu'on les frappe. Les touches modificatrices ne génèrent pas d'événement lorsqu'elles ne sont pas accompagnées de la frappe d'une touche non-modificatrice. On les appelle ainsi car elles modifient totalement la signifacation d'une touche:

La touche "**v**".
Avec Maj. Verrouillée, ou Majuscules: "**V**".
Avec la touche "Option:"√".
Avec les touches "Option" et Majuscules:"◊".
Avec la touche "Commande": **COLLE** le contenu du presse-papiers à l'emplacement du curseur (certainement un point d'insertion).

Afin de savoir si une quelconque touche modificatrice a été frappée au moment de la génération d'un événement, on interroge le champ "modifier" dans l'event record.

L'EM en personne génére un événement de type autoKey si on appuie sur une touche pendant quelques secondes, si les conditions suivantes sont remplies:
- il n'y a aucun événement de priorité supérieure en attente,
- aucune touche modificatrice n'a été frappée simultanément,
- le masque d'événement permet l'événement de type autoKey,
- au moins 16 ticks se sont écoulés
- un certain intervalle de temps, comparé à une variable (dont vous fixez la valeur, délai entre deux frappes, dans le Tableau de Bord), s'est écoulé depuis le dernier événement de type clavier.

STRUCTURE DES EVENEMENTS

Voici la structure de l'EventRecord telle que nous l'offre la Toolbox:

```
type
EventRecord=record
    what:integer;
```

```
        message:LongInt;
        when:LongInt;
        where:Point;
        modifiers:integer;
    end;
```

Soit, dans un programme:

```
var
    myevt: eventrecord;
```

On peut savoir le type d'événement - myevt.**what**, un entier de Ø à 15 - le moment où s'est produit l'événement - myevt.**when**, en ticks - où il s'est produit (en coordonnées globales), sous forme de point.

L'information contenue dans le champ "**message**" dépend du type d'événement. Le champ "**modifiers**" renseigne sur la ou les touches frappées au moment de l'événement.

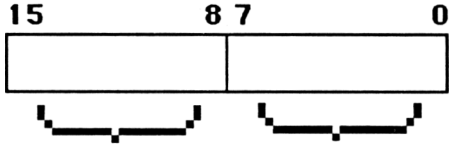
Comme d'habitude, dans l'EM comme dans d'autres Managers, on trouve un jeu de constantes qui permettent de tester les événements de manière cohérente. Le champ "**what**" peut prendre une des valeurs suivantes:

```
const
    nullEvent    =   0;  { null Event }
    mouseDown    =   1;  { clic sur la souris }
    mouseUp      =   2;  { bouton de la souris relâchée }
    keyDown      =   3;  { touche appuyée }
    keyUp        =   4;  { touche relâchée }
    autoKey      =   5;  { touche maintenue appuyée }
    updateEvt    =   6;  { mise à jour }
    diskEvt      =   7;  { insertion de disquette }
    activateEvt  =   8;  { sélection de fenêtre }
    abortEvt     =   9;  { annulation }
    networkEvt   =  10;  { de type réseau - AppleTalk }
    driverEvt    =  11;  { E/S }
    app1Evt      =  12;  { défini dans l'application }
    app2Evt      =  13;  { défini dans l'application }
    app3Evt      =  14;  { défini dans l'application }
    app4Evt      =  15;  { défini dans l'application }
```

Le champ "message"

L'information contenue dans ce champ, nous le savons, est spécifique à la nature de l'événement. Dans le cas d'un

événement de type "keydown", le champ "message" contiendrait:



Format du champ "message"
des keyboard events

L'octet de droite contient le champ "message" modulo 256, donnant le code ASCII du caractère. Quant à l'octet de gauche, sa valeur indique le code de la touche. On se sert de celui-ci dans certaines applications du type traitement de texte - ou pour définir des assignations de notes aux touches d'un clavier. On a accès à ces valeurs en se servant de **loword** et **hiword**.

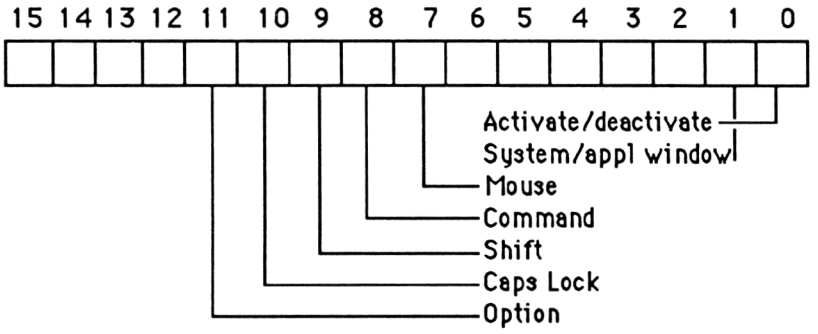
En ce qui concerne les diskEvt, le champ "message" donne le numéro de drive concerné. "1" pour le disque interne, "2" pour le disque externe, "3" pour le RamDisk, "4" pour le HardDisk du Mac XL. Si, pour une raison ou une autre, la lecture du disque ne pouvait s'effectuer, l'OS place un code d'erreur dans l'octet no 3. La raison peut en être: une disquette détruite, non initialisée,...

Dans le cas des **updateEvt** et des **activateEvt**, tous deux liés aux fenêtres, le champ message contient le **pointeur** sur la fenêtre concernée. Il ne contiendra rien de significatif en cas de **mouseUp**, **mouseDown** et **nullEvent**, mais contiendra la valeur de la touche qui annule une opération en cas **d'abortEvt.**

Enfin, pour les événements définis dans l'application, le champ "message" contient ce que le développeur de logiciels décide. Cela peut être un pointeur, un flag, une constante, un handle,...

Le champ "modifiers"

Ce champ contient l'état du bouton et celui de la touche "modifier" au moment du 'timbrage' de l'événement. On peur résumer ainsi: tout bit à 1 signifie "touche enfoncée". Les bits non assignés de la Fig. suivante sont tout simplement inutilisés.



le champ "modifiers"

En cas d'activation d'une fenêtre, celle qui était active est d'abord dé-activée. Vous savez ce que signifient ces termes, nous les avons vu en parcourant le Finder, au début du chapitre concernant les ressources. Une fenêtre dé-activée pert le grisé de ses contrôles, etc.

Ensuite seulement, la fenêtre activée devient réellement active. C'est à présent dans celle-ci que l'on peut dessiner, entrer du texte, ou toute opération spécifique à l'application.

Ouvrez deux fenêtres et créez une fenêtre active plus petite que celle qu'elle recouvre. Cliquez sur la barre de menu de la fenêtre *inactive* **sans relâcher le bouton de la souris**.

Jusqu'à présent, le contenu de la fenêtre qui était inactive n'a pas changé. Un événement de type **activate** a été généré, puisque la fenêtre inactive est devenue active, et vice-versa, mais jusqu'à présent, il n'y a pas eu de mise à jour du contenu de la fenêtre.

A présent, relâchez le bouton de la souris: et voyez s'opérer la mise à jour de la fenêtre.

Le bit Ø du champ "modifiers" est mis à "1" si une fenêtre est dé-activée. Il est mis à "Ø" si la fenêtre est dé-activée. Le champ "message", lui, contiendra le pointeur sur la fenêtre concernée. On peut à présent traiter l'information.

On rencontre plusieurs types de fenêtres:

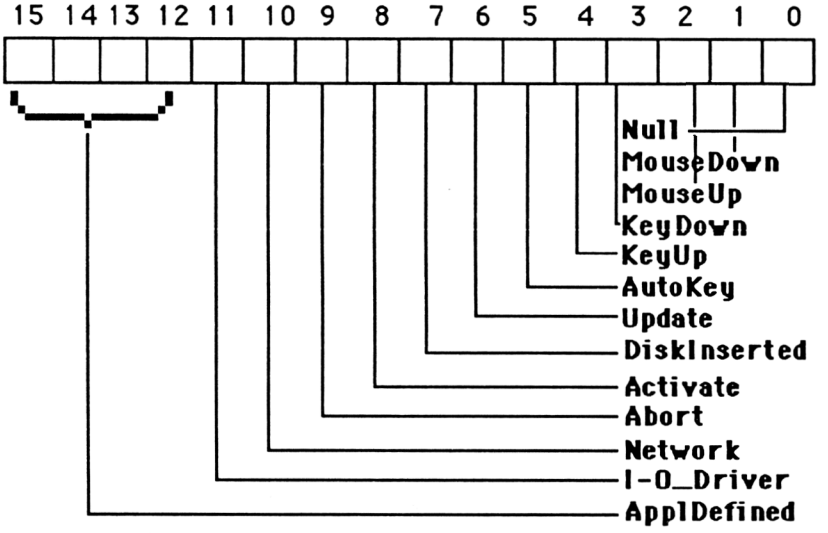
- La fenêtre d'applications, créée par l'application, avec ou sans contrôles, dans laquelle on effectue ce pour quoi

l'application a été écrite,

- La fenêtre système, des accessoires de bureau,

Le bit 1 du champ "modifiers" est mis à "1" si une seule des fenêtres d'un processus dé-activation/activation appartient à un programme d'applications. Si les deux fenêtres sont de même type, le bit 1 est mis à "Ø". Les bits 2 à 6 et 12 à 15 sont inutilisés.

Masques



Carnaval: Le masque d'événements

A cause de la masse d'informations fournie par l'EventRecord, et de la quantité de type d'événements, on peut vouloir restreindre le nombre des événements en ne sélectionnant que ceux dont on croit avoir besoin.

Pour ce faire, il y a l'"EventMask", qui est programmable. C'est un entier qui permet de sélectionner les événements "intéressants". Voici les constantes que l'on peut utiliser:

```
const

    nullMask     =   1;
    mDownMask =   2;
    mUpMask      =   4;
```

```
keyDownMask=   8;
keyUpMask  =   16;
autokeyMask =  32;
updateMask  =  64;
diskMask    =  128;
activMask   =  256;
abortMask   =  512;
networkMask =  1024;
driverMask  =  2048;
app1Mask    =  4096;
app2Mask    =  8192;
app3Mask    =  16384;
app4Mask    =  -32768; { complément à 2 }
```

Le masque fournit aussi une valeur mettant tous les bits du masque à 1, pour traiter tous les types d'événements: everyEvent = -1;

Pour utiliser ce masque et lui donner la valeur voulue, il est des **utilitaires** qui permettent d'assigner des valeurs à un entier: la fonction AandB, vue au chapitre I et légèrement modifiée pourrait faire l'affaire. **BitTst**, **BitClr** et **BitSet** (baseAddr + déplacement au niveau du bit) pourraient s'en sortir. L'opérande baseAddr, c'est le bit Ø. C'est l'inverse de la convention utilisée sur ce processeur!

Mais pour donner une valeur à ce masque, on peut encore et surtout jouer directement avec les noms attribués aux constantes. Exemple: si, dans une boucle de traitement locale, vous voulez sélectionner tous les événements de type clavier, vous écrirez:

```
MyMask:= keyDownMask + keyUpMask + autoKeyMask;
```

et ainsi de suite. Dans cet exemple, MyMask aurait une valeur de: 8 +16 + 32 = 56.

L'EM est aussi constitué de certaines fonctions et procédures, que nous abordons aussitôt, en commençant par la plus importante:

GetNextEvent

```
function GetNextEvent(eventMask:integer;
                      var theEvent:EventRecord): boolean;
```

GetNextEvent enlève un événement de la file d'attente et place ses caractéristiques dans "theEvent", un eventrecord défini

comme une variable de l'application.

APPELS DIVERS

Il est souvent utile de savoir où dans une fenêtre un utilisateur a cliqué, de manière à prendre l'action correspondante. C'est la routine du Window Manager FindWindow qui s'en chargera. Il faut donc untiliser "*InitWindows*" avant de se servir de l'EM.

Si vous voulez vider la file d'attente des événements, utilisez "**FlushEvents**" - en général avant la boucle d'événements.

La fonction **GetDblTime** retourne dans un Longint le temps maximum (en ticks) séparant deux mouseDown.

GetMouse

```
procedure    GetMouse(var MouseLoc:Point);
```

Appelez la procédure "GetMouse" pour obtenir à tout moment la position de la souris, en coordonnées locales au Grafport.

GetKeys

```
procedure    GetKeys(var theKeys:keyMap);
```

Voici ce qu'est un keyMap:

```
type
    keyMap =    packed array[0..127] of boolean;
```

"GetKeys" vous donne l'état du clavier, et des touches enfoncées au moment de l'appel.

Button

function Button : boolean; on l'utilise pour savoir si le bouton de la souris est enfoncé ou non.

StillDown

```
function StillDown:boolean;
```

Si, après un événement de type mousedown, on veut savoir si quelque chose est transporté (par exemple, l'ascenseur, dans une barre de contrôle), on utilise StillDown, qui retournera TRUE si le bouton est encore maintenu, et qu'il n'y a pas d'événement de type mouseUp ou mouseDown dans la file d'attente.

On peut l'utiliser dans une boucle: **while** not **StillDown** do...

WaitMouseUp

function WaitMouseUp:boolean;

Après un mouseDown, WaitMouseUp retourne TRUE si le bouton a toujours été maintenu depuis le mouseDown du début.

Afin de conserver la trace de la souris, le bouton maintenu appuyé, on peut se servir de *StillDown* ou de *WaitMouseUp*.

EventAvail

EventAvail utilise les mêmes paramètres que GetNextEvent mais ne retire pas l'événement de la file d'attente (s'il correspond à un des événements filtrés par le masque).

PostEvent

procedure PostEvent(eventcode:**integer**; eventMsg:**LongInt**);

Si vous définissez des événements (app1Evt .. app4Evt), vous les placez dans la file d'événements en les 'postant'.

PostEvent permet aussi de poster un événement déjà retiré de la file d'attente des événements par un GetNextEvent. En ce cas, faites attention à ne pas poster n'importe quoi: des paramètres comme la position de la souris, l'instant de l'événement, etc. peuvent avoir changé depuis que l'événement a été oté de la file.

FlushEvents

procedure FlushEvents(eventMask,stopMask:**integer**);

Si le stopMask est à Ø, FlushEvents retire tous les événements du type(s) spécifié dans l'eventMask. FlushEvents(everyEvent,Ø) nettoiera toute la file d'attente.

TickCount

function TickCount:**LongInt;**

Cette fonction retourne le nombre de ticks (1/60è de seconde) écoulés depuis le dernier chargement du système.

SetEventMask

procedure SetEventMask(theMask:**integer**);

Vous rappelez-vous le calcul effectué après la sélection "keydownMask + keyUpMask..." dont le résultat était 56? Voici une autre manière de fixer une valeur au masque d'événement.

SetEventMask(56); serait revenu au même que la manière de procéder que nous avons vue pour sélectionner les événements de type "clavier".

Les routines qui suivent sont d'usage plus restreint. Elles concernent l'OS Event Manager:

OSEventAvail, et GetOSEvent.

Nous n'entrerons pas dans les détails; mais retenez cette caractéristique intéressante du Mac, à savoir le *journaling*. Les événements peuvent être enregistrés puis, rejoués. Excellent pour les démos, ou le travail automatique: si vous enregistrez une série de commandes, vous pourrez les faires s'exécuter. Voilà comment fonctionne un petit utilitaire du domaine public.

Certaines routines du WindowManager peuvent être évoquées ici, puisque elles concernent la boucle d'événements: c'est le cas de la fonction FindWindow. Lorsqu'un mousedown se produit, il est intéressant de savoir où il s'est produit. La Toolbox offre quelques constantes; les voici, leurs noms sont suffisemment parlants:

```
const
    inDesk = 0;
    inMenuBar = 1;     { dans la barre de menus }
    inSysWindow  = 2; { dans fenêtre d'access. de bureau }
    inContent = 3;     { dans le contenu d'une fenêtre }
    inDrag = 4;        { dans la barre de titre }
    inGrow = 5;        { dans l'icone de dimensionnement }
```

inGoAway = 6; { dans la boîte de fermeture }

En résumé: toute l'interactivité d'un programme Mac vient du fait qu'il interroge constamment les types d'événements qui peuvent s'être produits. Ce, dans une boucle, puisque c'est le moyen le plus facile. Le programme sait donc quel événement s'est produit, et le traite:

a) selon la partie de l'écran où il s'est produit,

b) la spécificité de l'application.

Echantillon de boucle d'événements:

```
If GetNextEvent(EveryEvent, theEvent) then

case myEvent.what of

    mouseDown:
    case FindWindow(theEvent.where,whichWindow) of

        inMenuBar:
          traiter les menus;
          lancer les tâches correspondantes, selon l'élément
          du menu sur lequel le bouton de la souris a été
          relâché.

        inSysWindow:
          dans une fenêtre système. c'est le système qui s'en
          charge d'un SystemClick, via les accessoires de
          bureau, ou d'autres.

        inDrag:
          dans la barre de titre de la fenêtre.
          si la fenêtre n'est pas active, la rendre active;
          sinon, déplacer la fenêtre.
          Une variante: la touche Commande enfoncée en
          même temps n'implique pas l'activation de la
          fenêtre, si celle-ci était inactive.

        inGoAway:
          dans la boîte de fermeture de la fenêtre:
          si la fenêtre n'est pas active, la rendre active;
```

fermer la fenêtre. En certaines occasion, on peut aussi quitter l'application, bien qu'il soit préférable de passer par la barre de menus et "Quitter".

inGrow:
dans la boîte de dimensionnement:
si la fenêtre n'est pas active, la rendre active;
re-dimensionner la fenêtre.

inContent:
du moins, dans la portion visible de la fenêtre
si la fenêtre n'est pas active, la rendre active;
traitement en fonction de l'application.

end; { case mouseDown }

keyDown,autoKey;
événement de type clavier.
il s'agit soit de texte à traiter, soit d'un 'raccourci';
il faut traiter cet événement en tenant compte du champ '*modifier*'.

activateEvent;
il faut traiter cet événement selon l'application: par exemple, s'il y a deux fenêtres de texte, il faudra veiller à sélectionner éventuellement le texte qui l'était avant qu'une fenêtre devienne inactive; et dé-sélectionner le texte sélectionné, s'il y en avait, sur la fenêtre qui devient inactive.

UpdateEvent;
événement de type 'mise à jour'
BeginUpdate; pour stopper la génération d'autres événements de type Update.
MISE A JOUR (texte ou graphisme;
EndUpdate; pour reprendre la génération d'événements de type Update.

end; { case myEvent.what }

Cette explication ressemblait à du pseudo-code. Voici la même en "C". Vous trouverez un exemple de boucle d'événements en Pascal dans les lignes de programme commentées:

```
do {
        SystemTask();   /* re-initialisation zones système */
        TEIdle(hTE);    /* re-init blink */
```

```
/*BOUCLE PRINCIPALE */
        If (GetNextEvent(_everyEvent,&myEvent) == TRUE)
            switch (myEvent.what) {
            case mouseDown:
                 code = FindWindow(myEvent.where.vh_l,&whichWindow) ;
                /* where */
                switch (code) {
                case inMenuBar:
                    DoCommand(MenuSelect(myEvent.where.vh_l)) ;
                    break ;
                case inSysWindow:
                    SystemClick(&myEvent,whichWindow) ;
                    /* Desk Accessory, access. bureau */
                    break ;
                case inDrag:
            DragWindow(whichWindow,myEvent.where.vh_l,&dragRect) ;
                    break ;
                case inGrow:
                /* pas traitée ici */
                    break ;
                case inContent:
                    if (whichWindow != FrontWindow())
                        SelectWindow(whichWindow) ;
                        /* génére un activate event */
                    else {
                        GlobalToLocal(&myEvent.where) ;
                        TEClick(myEvent.where.vh_l,FALSE,hTE) ;
                    }
                    break ;
                }
                break ;

            case keyDown:
            case autoKey:
                if (theWindow == FrontWindow()) {
                    keychar = myEvent.message % 256 ;
                    TEKey(keychar,hTE) ;
                }
                break ;

            case activateEvt:
                if ((myEvent.modifiers & 01) == 1)
                    TEActivate(hTE) ;
                else
                    TEDeactivate(hTE) ;
                break ;
            case updateEvt:
```

```
                SetPort(theWindow) ;
                BeginUpdate(theWindow) ;
                TEUpdate(&theWindow->portRect,hTE) ;
                EndUpdate(theWindow) ;
            }
    }while (doneFlag == FALSE) ;
}
```

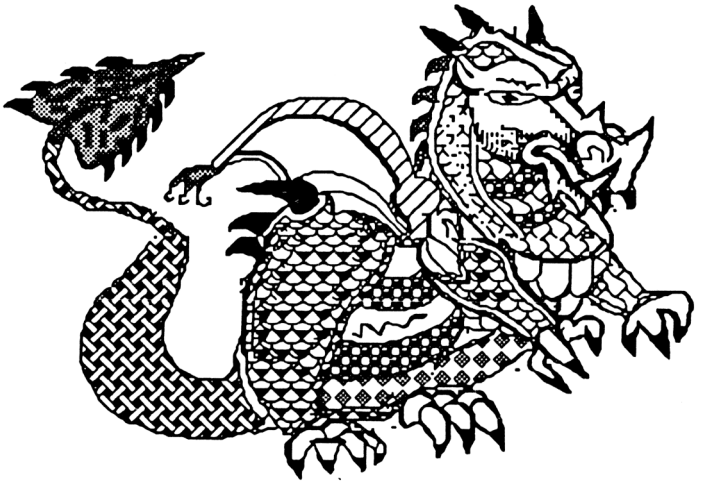
Tous les types d'événements ne sont pas nécessairement traités. Voyez-vous ici un quelconque **MouseUp**? Il est même des événements qui peuvent être traités dans une portion de code qui ne concerne que l'*intérieur* d'une fenêtre (incontent).

Vous remarquez qu'en Pascal et en "C", ce sont les mêmes noms qui sont utilisés, et qui testent les mêmes *codes retour* (cad inMenuBar, inSysWindow, etc.). Les mêmes noms pour effectuer les mêmes tâches (cad FindWindow, afin de savoir dans quelle portion de l'écran l'utilisateur a cliqué, SelectWindow, GlobaltoLocal, etc.).

Voici un autre exemple de boucle d'événements écrite en Forth. Elle vous montre où vous avez cliqué:

```
( Exemple de boucle d'événements )
: typem  ( -- )
  begin do.events ( do.events monitors events and automatically)
     ( executes words depending on the event.  A value is left )
     ( on the stack reporting what event has occured.  Note Any)
     ( event that activates another window will deactivate the )
     ( front window and activate the new window, ending typem  )

     case mouse.down  of cr ." mouse down" endof
        mouse.up    of cr ." mouse up" endof
        in.size.box of cr ." in size box" endof
        in.drag.box of cr ." in drag box" endof
        key.down    of cr ." pressed " key emit endof
        disk.event  of cr ." insert event" endof
     endcase
  again ;
```

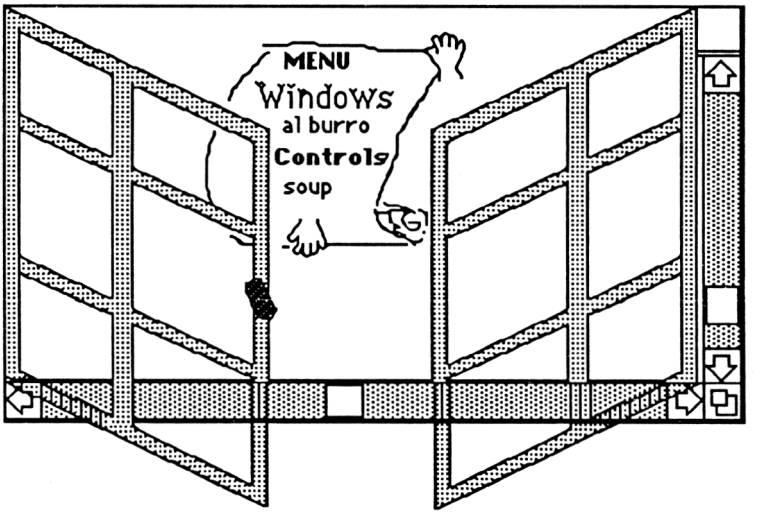
Celle-ci ne testait pas la barre de menus, mais montre la simplicité d'une boucle d'événements. Lorsque vous avez une boucle d'événements parfaitement testée, elle peut vous servir pour beaucoup de programmes à venir. Les spécifications peuvent varier, mais le principe reste le même.



Comprendre le Mac?

CHAPITRE IX

FENETRES ET CONTROLES



CONTENUS DES FENETRES

On n'écrit pas sur un bureau. On prend une feuille de papier pour y dessiner, ou y écrire. En bref, pour y traver un graphisme, formel ou non.

Le Window Manager (**WM**) garde la trace des fenêtres qui se chevauchent grâce à une liste chaînée.

Toute l'information relative à une fenêtre se trouve dans un Window Record, qui contient un GrafPort. Si une fenêtre a été

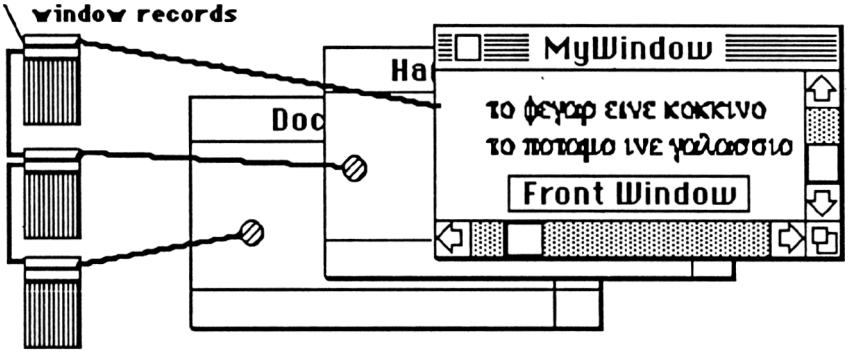
définie dans un fichier ressource, on l'appelle en utilisant GetNewWindow; le fichier des ressources contiendra déjà toute l'information nécessaire pour afficher la fenêtre.

Autrement, il faudra utiliser 'NewWindow', en passant des paramètres (dimensions, etc.) depuis l'*intérieur* du programme. Il en sera de même avec les dialogues.

Une application crée des fenêtres d'application. Une fenêtre qui affiche un accessoire de bureau est appelée fenêtre système. Les fenêtres sytème sont gérées différemment des fenêtres d'application.

Certaines formes de fenêtres sont pré-définies. Une fenêtre a toujours un **cadre** et un **contenu**. Hors du cadre, on trouve:

- une barre de titre,
- une boîte de fermeture,
- des contrôles (limités à la barre de défilement et l'icone de dimensionnement).



Si, sur votre écran, vous avez plusieurs fenêtres présentes simultanément, celle qui se trouve au premier plan est dite "active". Les deux autres fenêtres montrées sur la figure précédente sont *inactives*. Un clic n'importe où sur une fenêtre inactive doit la rendre active - par SelectWindow (à moins que l'application ne demande le contraire - c'est rare).

Toute entrée de texte ou de dessin, ou tout affichage s'effectue sur la fenêtre **active**.

Le contenu de la fenêtre, c'est le champ portRect du Grafport.

L'application doit se charger du contenu. L'action du WM est limitée au tracé de la fenêtre et de la barred de titre.

Le GrafPort du WindowRecord sert de port au WM.

Le cadre et le contenu peuvent être considérés comme des **régions**. Celle qui entoure l'ensemble, care et contenu, s'appelle la **"structure region"**.

De plus, si la fenêtre comporte une boîte de fermeture, elle aura une **go-away region** à l'intérieur de la barre de titre, éventuellement une **drag region**, et une **grow region**. Cette dernière fait partie du contenu. Cette dernière est utilisée pour redimensionner la fenêtre.

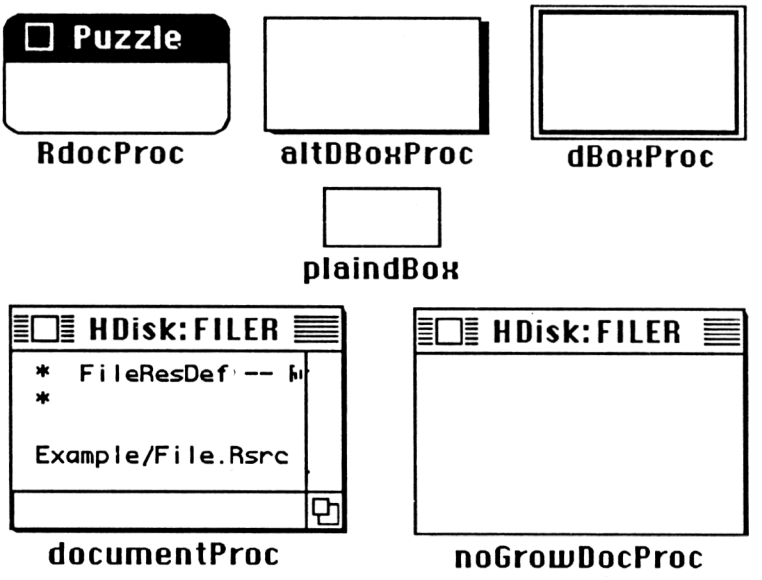
Les zones d'une fenêtre doivent être retracées lorsqu'une fenêtre devient active, et que le contenu de celle-ci était recouvert par la précédente fenêtre active.

Ces zones sont prises en compte par le WM. On dit qu'elles forment l'**update region**. Il faudra utiliser des zones du WM pour stocker des portions de fenêtre dans l'update region.

TOUTES CES FENETRES...

Le WM fournit quelques constantes pour définir les formes de fenêtres standard.

Sur la page suivante, vous pouvez voir les forme pré-définies qui vous sont offertes.



Vous avez ainsi un très large choix de fenêtres. Vous avez même la possibilité de choisir la coubure du rDocProc, en ajoutant un chiffre compris entre 1 et 7 à la rDocProc.

Car il s'agit de procédures de tracé de cadre de fenêtre. Et vous pouvez définire les vôtres.

const

```
documentProc    =   0;  { standard }
dBoxProc        =   1;  { modal dialog }
plainDBox       =   2;  { vide }
altDBoxProc     =   3;  { vide + ombrée }
noGrowDocProc   =   4;  { comme la fenêtre }
                        { de MacPaint }
rDocProc        =  16;  { coins arrondis }
```

Examinez la ressource de type WIND, au chapitre VI. Le champ windDefProcs du window record spécifie la procédure qui tracera le cadre choisi.

Cette procédure, tout comme une PICTure (vous savez ce qu'est une picture depuis le chapitre V) trace la fenêtre telle qu'elle se présente à son premier affichage.

Les windDefProcs tracent aussi la boîte de dimensionnement dans le contenu de la fenêtre, retournent un handle sur la region où s'est produit un clic, et calculent finalement les strucRgn et contRgn. Ces windDefProcs ont vraiment de quoi faire...

Voici le Window Record, tel que décrit dans le Toolintf:

LE WINDOW RECORD

```
type
    WindowPtr       =GrafPtr;
    WindowRecord    =    record
        port            : GrafPort;
        windowKind      : integer; { classe de la fenêtre }
                          { fenêtre d'application, système }
                          { dialogue,  }
        visible         : boolean; { TRUE: visible }
        hilited         : boolean; { TRUE: barre de titre 'hilit' }
        goAwayFlag      : boolean; { TRUE: boîte de fremeture }
                                   { 'hilit' pour documentProc, }
                                   { noGrowDocProc  et }
                                   { rDocProc }
        spareFlag       : boolean;
        strucRgn        : RgnHandle; { Rgn Handles }
        contRgn         : RgnHandle; { en coordonnées }
        updateRgn       :RgnHandle;  { globales }
        windowDefProc: Handle;    { Handle sur la window }
                          { fonction de définition de }
                          { la fenêtre }
        dataHandle      :Handle;
        titleHandle     :StringHandle; { Handlesur }
                          { une STR255, s'il y a un titre }
        titleWidth      :integer; { largeur du titre en pixels }
        controlList     :Handle; {handle sur le CDEF évent. }
                               { Le Control Mgr maintient la }
                               { control list, tête de liste }
                               { de la liste chaînée }
                               { des control records }
        nextWindow      : WindowPeek; { un pointeur }
    { liste chaînée de toutes les fenêtres, ordonnées }
    { depuis le premier plan, à l'arrière plan }
    { la dernière est à NIL }
        windowPic       :PicHandle; { Handle sur une évent. }
                               { picture QuickDraw du }
                               { contenu de la fenêtre }
```

```
        refCon          :LongInt;{ ce champ vous appartient }
{ pour y placer ce que vous voulez - pointeur, handle,
  numéro...}
      end;
```

Le même en "C":

```
# define boolean char
.....................
main()
déclaration des variables de travail
.....................
struct windowRec
{
    grafport        port;
    short           windowkind;
    boolean         visible;
    boolean         goawayflag;
    boolean         spareflag;
    rgnhandle       strucrgn;
    rgnhandle       contrgn;
    rgnhandle       updatergn;
    handle          windowdefproc;
    handle          datahandle;
    handle          titlehandle;
    short           titlewidth;
    handle          controllist;
    struct          windowrec * nextwindow;
    pichandle        windowpic
    long            refcon
    };
.....................
```

Notez bien:

Les champs visible et highlighted flags (deux booléens) sont mis à to "255" au lieu de "1". On les testera en "NOT FALSE", ou "nonzero".

EVENEMENTS ET FENETRES

Certaines routines du WM modifient l'état d'une fenêtre, la faisant passer d'active à inactive, générant des événements de type **activate/deactivate**. Alors, le pointeur sur la fenêtre est passé au champ 'message' de l'eventrecord. De même, le bit

'modifier' voit son activeFlag (bit Ø) mis à "1" si la fenêtre est activée. Si la fenêtre change de type (windowkind), cad passe de fenêtre d'application (ex-active) à fenêtre système, ou vice-versa, le changeFlag du champ 'modifier' est mis à "1" (bit 1).

Nous savons depuis le chapitre VIII que les événements de type activate arrivent par paire, le deactivate généralement en premier. Ces événements ont la priorité la plus élevée. Un seul activate est généré s'il n'y a qu'une fenêtre sur l'écran.

Les événements qui se produisent dans un dialogue ou une alerte sont pris en compte par le Dialog Manager.

AFFICHAGE DE FENETRES / UPDATE EVENTS

Les regions qui requièrent une mise à jour seront traitées lorsque se produira l'update event. Celui-ci est généré dès qu'une update region n'est pas vide.

Afin de traiter une mise à jour, il faut d'abord utiliser:

procedure BeginUpdate(theWindow:WindowPtr);

qui calcule l'intersection de deux regions: la visRgn de la fenêtre et l'update region. Le résultat est ce qui devient la **nouvelle** visible Region, et l'update region est vidée. L'update event est annulé (et ne sera plus pris en compte dans la boucle). Cette procédure empêche la génération d'autres événements de type update. Si on l'oubliait, le fait de ré-afficher générerait un autre update event.

Ensuite, il faut dessiner la visRgn, le contenu de la fenêtre. Son rectangle, c'est le champ bBox. Vous pouvez utiliser la routine **PtInRect** de QuickDraw pour vérifier l'intersection avec un rectangle:

```
function PtinRect(pt:Point; r:Rect): boolean;
{ retourne TRUE si le point est à l'intérieur du rectangle }

procedure SetRect(var r: Rect; left, top, right, bottom: integer);
```

On peut décider de ne pas afficher le complément visible, qui

correspond à l'update region, et de retracer **toute** la fenêtre. La visRgn peut ne pas être un rectangle (c'est une region). Donc, la manière de procéder dépend de votre bon vouloir.

Les procédures suivantes (QuickDraw) vous permettent de savoir si un point ou un rectangle sont à l'intérieur d'une region:

```
function PtinRgn(pt: Point; rgn: RgnHandle):boolean;
{ retourne TRUE si le point se trouve dans la region }

function RectinRgn(r: Rect; rgn: RgnHandle):boolean;
{ retourne TRUE si le rectangle se trouve dans la region }
```

Et enfin, le ré-affichage se terminera par:

```
procedure EndUpdate(theWindow: WindowPtr);
```

qui vide l'update region. Et la génération de nouveaux événements de type update peut reprendre. Cependant, après quelques essais, nous avons conclu qu'il était souvent préférable d'utiliser un handle sur une picture (dans le champ PicHandle du window record), et de forcer le réaffichage avec la picture comme source, ce qui va plus vite que le procédé "classique" mentionné ci-dessus.

Si vous voulez garder la trace du contenu de chacune de vos fenêtres, et créer un update correspondant aux portions de fenêtre rajoutées, utilisez **InvalRect** ou **InvalRgn**. Cet update déclenchera la mise à jour dans la boucle d'événement.

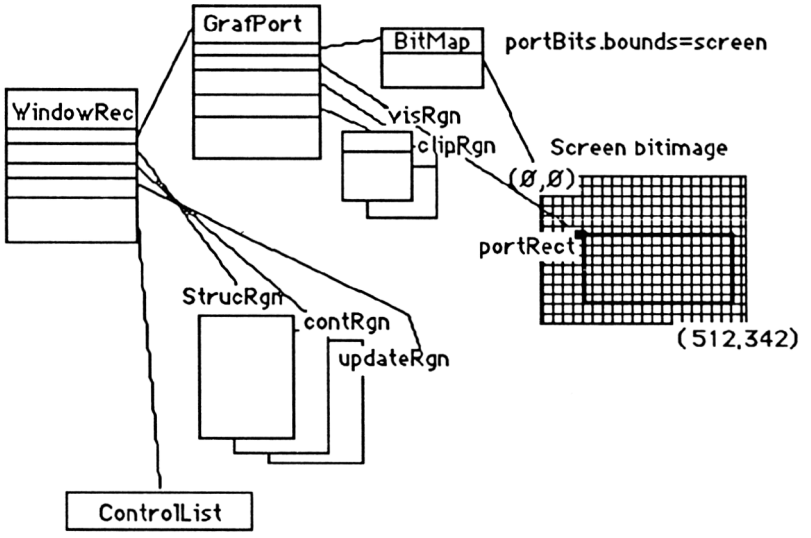
Un exemple simple de mise à jour de fenêtre pourrait ressembler à:

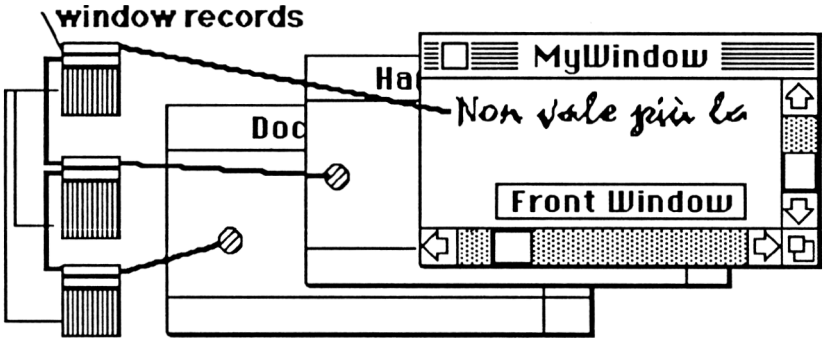
```
BeginUpdate(MyWindow);
EraseRect(MyWindow^.portrect); { effacement de }
                               { toute la fenêtre }
DrawGrowIcon(MyWindow);        { dimensionnement }
DrawControls(MyWindow);        { réaffich. des contrôles }
TEUpdate(pRect,hTE);           { m.à.j. du texte dans  pRect }
                               { un rectangle initialisé au }
                               { début de l'application }
                               { hTE: handle s/ le texte }
EndUpdate(MyWindow);           { c'est tout }
```

AU TOTAL

Toute mise à jour implique un réaffichage, même si le contenu de la fenêtre n'a pas changé.

C'est le contenu d'un WindowRecord qui apparaît dans la figure ci-dessous:





liste chaînée avec chaînage arrière vers le windowrecord de la fenêtre active

ROUTINES DU WINDOW MANAGER

Nous avons déjà vu l'utilisation d'**InitWindows** au chapitre VII. Cette procédure n'a pas de paramètre, et est écrite après InitGraf et InitFonts.

NewWindow permet de créer une fenêtre depuis un programme sans qu'il y ait de ressource;

GetNewWindow va dans le fichier des ressources chercher la ressource de type WIND dont le numéro est spécifié, et l'affiche.

FindWindow dit dans quelle partie de la fenêtre s'est produit le clic de la souris.

Selon la réponse, vous suivrez le chemin proposé dans l'organigramme qui suit:

InContent / d'une fenêtre inactive:

SelectWindow rend la fenêtre active.

/ d'une fenêtre active:

Le traitement dépend du type de fenêtre: **SystemClick**, dans une accessoire de bureau, réinitialise certaines zones du système, restaure l'horloge...

Si une fenêtre contient des contrôles du type 'boîte de dimensionnement', et qu'il vous faut re-dessiner le contenu, vous devez savoir si cette fenêtre est active ou non. Si elle est active, **DrawGrowIcon** dessinera l'icone à sa place.

inDrag: l'utilisateur a maintenu le bouton de la souris appuyé lorsque le curseur se trouvait sur la barre de titre.

Alors, **DragWindow** amènera la fenêtre à son nouvel emplacement (lorsque le bouton de la souris est relâché), et rendra la fenêtre active, sauf si la touche "Commande" est enfoncée.

inGrow: l'utilisateur a voulu modifier la taille de la fenêtre. Un appel à **GrowWindow** affichera à l'écran une ligne de pointillés montrant le changement de taille de la fenêtre. Puis, **SizeWindow** modifiera la taille de la fenêtre (lorsque le bouton de la souris sera relâché).

inGoAway indique le fait que l'utilisateur a cliqué dans la boîte de fermeture de la fenêtre. En ce cas, **TrackGoAway** se chargera des modifications visibles qui s'opèrent sur cette region.

Si le bouton de la souris est relâché **hors** de la region, on n'a rien à faire. Sinon, appelez **CloseWindow** ou **DisposeWindow**, afin de libérer la place.

Autres routines du WM

HideWindow fera temporairement disparaître la fenêtre, qui réapparaîtra d'un coup de **ShowWindow**.

Afin de savoir si une fenêtre est active ou non, **FrontWindow** vous donnera le windowPtr de la fenêtre active au moment où vous appelez cette fonction. Ce, afin de le comparer à la valeur du pointeur que vous voulez tester.

Si vous voulez stocker ou récupérer un handle sur une PICTure dans un windowrecord, utilisez **SetWindowPic**, ou **GetWindowPic.**

GetWTitle retournera le titre, dans une STR255. Pour placer un titre, utilisez **SetWTitle** qui placera le contenu d'une STR255 dans la barre de titre, remplaçant éventuellement ce qu'il y avait

auparavant.

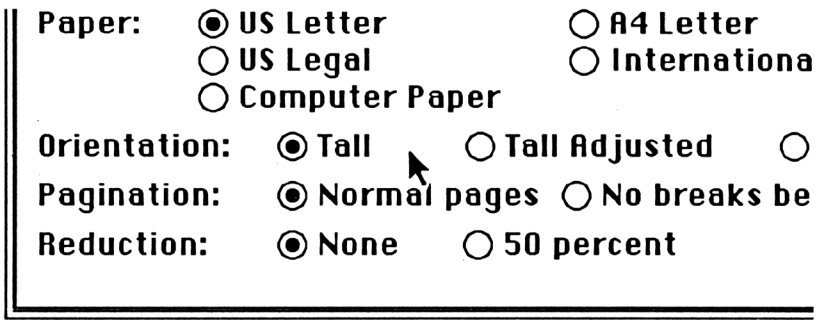
Vous pourrez trouver d'autres routines du WM dans l'Appendice Window Manager, en fin de ce livre. Elles sont peu utilisée 'dans la vie courante'...Nous ne les commentons pas.

Laissons à présent le Control Manager parler.

LES CONTROLES

Ce sont des éléments de graphisme, avec ou sans texte, qui fournissent une information au programme. Cette information est nouvelle, ou modifie des données entrées précédemment. Certains contrôles sont pré-définis. Mais vous pouvez créer les vôtres. Voici les contrôles standard:

- "Boutons Radio" (Radio-buttons). Ceux que l'on utilise dans les choix multiples:



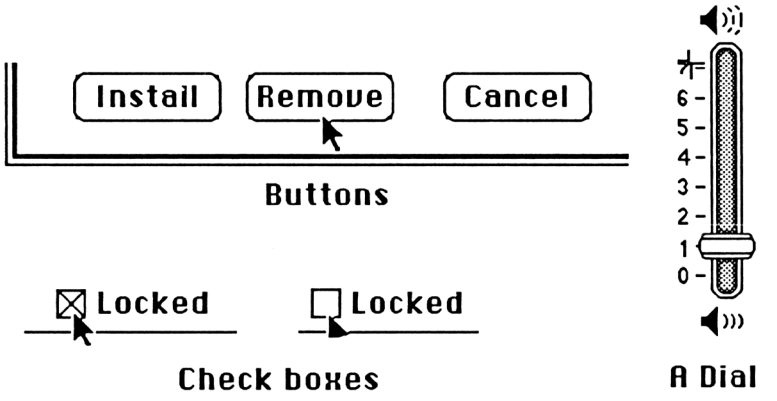
Radio-Buttons

Au développeur de gérer les choix offerts à l'utilisateur. Un choix peut en exclure d'autres.

- "Boutons" (Buttons). Ces rectangles arrondis contiennent d'habitude un message.

- "Carrés" (Check Boxes). Ils permettent le passage d'un

choix à un autre. Du point de vue du Control Manager (CM), ils sont identiques, sauf pour leur forme respective.
La figure ci-dessous vous les présente tous:



D'autres types de contrôles fournissent une indication visuelle. Le "dial" ci-dessus (que vous reconnaissez: nous l'avons pris dans le tableau de bord) ne fait **pas** partie des contrôles pré-définis. Il doit être dessiné.

N'oublions pas ces contrôles que sont les **barres de défilement**, avec l'ascenseur (thumb).

Tout comme les fenêtres, les dialogues, les menus, les polices de caractères, un contrôle peut être défini à l'intérieur d'un programme, ou dans un fichier des ressources.

Une barre de défilement (scroll bar) appartient à une fenêtre. On peut fort bien définir un même contrôle pour plusieurs fenêtres, mais gare à ne pas s'emmêler!

D'ailleurs, une des premières fonctions du CM assigne la "scroll bar" à la fenêtre dans laquelle elle évoluera.

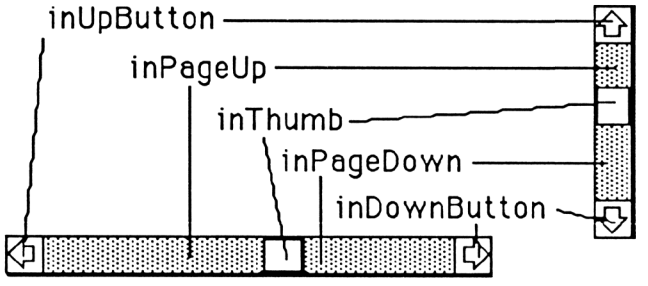
Une partie de la définition du contrôle est vouée à l'assignation de trois valeurs (par défaut, minimale et maximale). Vous comprenez donc la **signification** du mot contrôle: il s'agit d'une zone d'une fenêtre ou d'un dialogue permettant à l'utilisateur de fixer et/ou de recueillir des informations dans un programme.

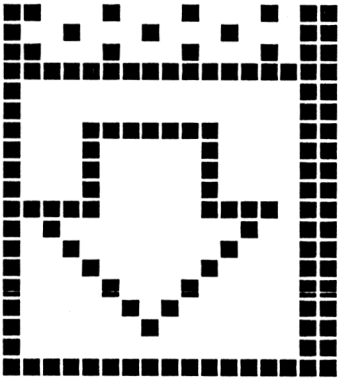
Chaque type de contrôle a son propre code. La ToolBox

fournit ici encore des constantes que l'on utilise après un **FindControl** afin de les comparer à la position de la souris dans un contrôle:

```
const

    inButton      =   10;  { bouton standard }
    inCheckBox    =   11;  { radio-button, ou Check Box }
    inUpButton    =   20;  { flèche supérieure }
    { ou droite d'une barre de défilement }
    inDownButton=    21;  { flèche inférieure }
    { ou gauche d'une barre de défilement }
    inPageUp      =   22;  { défilement d'une page }
                           { vers le haut }
    inPageDown    =   23;  { défilement d'une page }
                           { vers le bas }
    inThumb       =   129;{ dans l'ascenseur }
```

La section du contrôle dans laquelle se produit le clic de la souris est connue sous le nom de **partcode**.





La Fig. ci-dessus montre une flèche: les 16 pixels occupés par la barre de défilement, et le début de la region **inPageDown**.

Ce contrôle est dit "actif"; il inclut une modification de l'affichage (devient noir si l'on le clique).

C'est le CM qui dessine le contrôle sur l'écran. L'information sur les contrôles provient des ressources, ou bien est fournie dans le programme par les ordres d'affichage des contrôles. La forme du contrôle est fournie par une procédure de dessin:

```
{ID des procédures de dessin de contrôles }

pushButProc   = 0;
checkBoxProc  = 1;
radioButProc  = 2;
scrollBarProc = 16;
```

Les noms sont significatifs pour qui parle anglais.

LE CONTROL RECORD

Toute l'information relative à un contrôle est contenue dans un...Control Record. C'est une structure de donnée **dynamique**, et on y a accès par un ControlHandle. Le CM n'a pas besoin de beaucoup de noms spécifiques. Ne vous tourmentez donc pas pour apprendre tout cela par coeur.

```
type
    ControlPtr          = ^ControlRecord;
    ControlHandle       = ^ControlPtr;

ControlRecord = packed record
  nextControl:      ControlHandle;
  contrlOwner:      WindowPtr;
  contrlRect:       Rect;
  contrlVis:        Byte;     { TRUE: visible }
  contrlHilite:     Byte;     {    Ø ≤ ControlHilite ≤ 255: }
                              { hilit du contrôle }
  contrlValue:      integer;  { fixé à Ø off, 1 on }
  contrlMin:        integer;  { valeur minimale du contrôle }
  contrlMax:        integer;  { valeur maximale du contrôle }
  contrlDefProc:    Handle;   { hdl sur la proc. de définition }
                              { du contrôle }
  contrlData:       Handle;
  contrlAction:     ProcPtr;  { ptr sur la proc. de défin. }
  contrlRfCon:      longint;  { val. réf. du contrôle }
  contrlTitle:      Str255;   { titre }
end; {ControlRecord}
```

Vous avez certainement remarqué le champ "nextControl", liste chaînée sur le controlRecord suivant. On trouve le début de la liste dans le champ "controlList" du WindowRecord. La fin de la liste est à NIL.

ControlOwner est un pointeur sur la fenêtre "maîtresse" du contrôle.

Le contrôle est délimité par le rectangle ContrlRect, défini dans les ressources (type CNTL), ou dans le programme.

ROUTINES DU CONTROL MANAGER

La fonction de définition de la forme d'un contrôle se trouve généralement dans les ressources de type CDEF (control

definition function). La plupart des contrôles, cependant (commencez par créer des contrôles standard) utilisent les CDEF du fichier système.

Lorsque des contrôles font partie d'une alerte ou d'un dialogue, le Dialog Manager les gérera partiellement.

La fonction:

```
GetNewControl(controlID:integer; owner:WindowPtr): ControlHandle;
```

affiche un contrôle si le contrôle a été défini dans le fichier des ressources. Sinon, c'est la fonction:

```
NewControl(curWindow:    windowPtr;
           boundsRect:   Rect;
           title:        Str255;
           visible:      boolean;
           value:        integer;
           min:          integer;
           max:          integer;
           contrlProc:   integer;
           refCon:       Longint): ControlHandle;
```

qui s'en chargera.

Les contrôles mentionnés dans les lignes qui précédent sont les barres de défilement. Nous donnerons des exemples de gestion d'autres types de contrôle, dans des dialogues.

```
DisposeControl(theControl: ControlHandle);
```

enlève un contrôle d'une fenêtre, et

```
KillControls(theWindow:WindowPtr);
```

les elève tous de la fenêtre.

Nous avons déjà mentionné la fonction:

```
FindControl(thePoint:        Point;
            theWindow:       WindowPtr;
            var theControl:  ControlHandle): integer;
```

Si l'entier retourné n'est pas à zéro, l'utilisateur a cliqué dans un contrôle. Les codes correspondant à **inPageUp**, **inThumb**,

etc. vous renseignent sur l'endroit où il a été cliqué (le part-code). **inCheckBox** fonctionne aussi pour les radioButtons. On utilise cette fonction après **FindWindow**, en séquence d'un **GetNextEvent**.

Après un FindControl, la fonction:

TrackControl(theControl: ControlHandle;
actionProc: ProcPtr;
thePoint: Point): integer;

suivra la position de la souris, et détectera un éventuel **mouseUp**.

Nous avons déjà mentionné la procédure:

DrawControls(theWindow: WindowPtr);

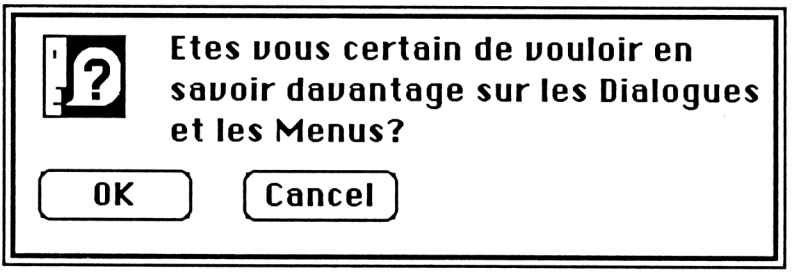
qu'on utilise par exemple durant le processus de mise à jour d'une fenêtre.

D'autres routines du CM ne sont pas aussi couramment utilisées. Citons-en quelques-unes GetCTitle, et SetCTitle, GetCtlValue, SetCtlValue...

A présent, une petite pause avant d'entamer... le dialogue.

CHAPITRE X

DIALOGUES ET MENUS...



POURQUOI DES DIALOGUES?

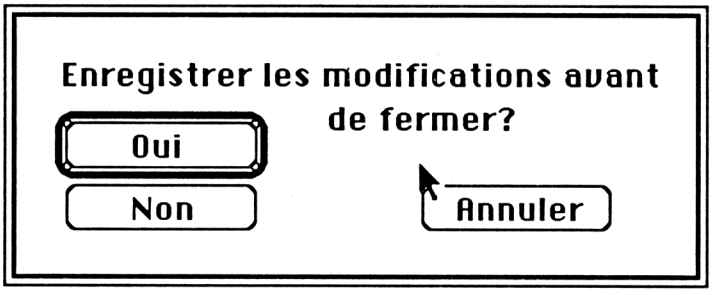
Afin d'être consistant avec sa propre philosophie, Mac se devait de saisir et récupérer les données de façon standard. Vous avez déjà vu des exemples de dialogues: **Enregistrer**, **Enregistrer sous...** dans MacWrite ou Paint. Ces boîtes vous permettent d'entrer du texte (le nom du fichier sous lequel sera sauvegardé un document).

Le texte contenu dans des dialogues peut être **statique** (comme celui présenté dans l'en-tête de ce Chapitre), ou **dynamique** (comme la boîte de texte présentée dans "Enregistrer").

Tous les dialogues, ainsi que les alertes, dialogues un peu spéciaux, requièrent une réponse. Dans d'autres machines, on

affichait un message avec une boucle de temporisation, et il disparaissait au bout de trois secondes. Pas sur le Mac: on s'assure que l'utilisateur a vu un certain message, et clique sur **SUITE**, ou **OK**. Par exemple, si une alerte affiche sur l'écran un message signifiant que le disque est plein et qu'on ne peut sauvegarder du texte, un utilisateur devra regarder dans ses disquettes et voir sur laquelle il peut effectuer sa sauvegarde. Tout ceci prend du temps. Il cliquera sur le **bouton** OK quand il sera **prêt**.

Un dialogue tel que celui présenté ci-dessous prend l'option présentée **par défaut** lorsqu'on frappe les touches Retour Chariot ou Enter. Ce type de dialogue, où **tout est bloqué** (Menus, etc...) est nommé **modal**. Or, le principe de la programmation du Mac est d'être a-modal.



Si vous développez des logiciels, laissez toujours l'alternative d'annuler. Offrez à l'utilisateur lorsqu'il est sur le point de faire quelquechose de dangereux (comme quitter sans avoir enregistré, par exemple).

Les dialogues et les alertes sont définis dans des ressources **DLOG**, **DITL** (**D**ialog **ITem** **L**ist), et **ALRT**.

Nous allons vous les présenter, ainsi que les éléments de programmation qui les appellent.

Type DLOG (36)
* Dialogue, purgeable (32) et pre-load (4); 32 + 4 = 36
,128
* No du dialogue
48 6 334 506

```
* position top left...
Visible GoAway
Visible, avec boîte de fermeture
16
* 16: cadre genre accessoires de bureau - puzzle.
0
* proc
128
* DITL associé

Type DITL
* le voici
  ,128(32)
4
* 4 items dans ce DITL
* item 1
BTNITEM ENABLED
* c'est un bouton
270 420 290 480
* sa position sur l'écran
SUITE
* le texte qu'il contient

* item 2
STATTEXT DISABLED
* texte statique. Disabled: si vous cliquez dessus,
* rien ne se produit. Vous ne pouvez pas
* entrer de texte
173 4 187 109
* sa position
Votre adresse :
* le texte contenu: votre adresse

* item 3
EDITTEXT ENABLED
* boîte d'entrée de texte. On peut éditer ce texte
173 114 187 476
* sa position
```

Il suffit de créer un "EditText" dans un dialogue, et il est alors possible d'entrer du texte en se servant du handle associé (nous l'étudierons bientôt). Si plusieurs EditText se trouvent dans un dialogue, on passe de l'un au suivant par la touche TAB.

La séquence d'instructions ci-dessous aura pour effet de vous montrer l'emploi de certaines routines du DM.

```
{-------------------------------------------------}
function dh(mydi: dialogptr; noitem: integer): handle;
{ récupération du handle sur le texte de l'EditText }

  var
   l: integer;
   hl: handle;
   myr: rect;

  begin
   getditem(mydi, noitem, l, hl, myr);
   dh := hl;
   end; {dh}

{-------------------------------------------------}
 procedure drawa;
   var
    i_hit, ind: integer;
    mydia, myport: dialogptr;

   begin
    getport(myport);
    { sauvegarde du port courant }
    mydia := getnewdialog(128, nil, pointer( - 1));
    { appel du DLOG }
    setport(mydia);
    { rendre ce port actif }
    setwtitle(mydia, 'SAISIE DE L"ADRESSE');
    { donnéer un titre }
    getditem(mydia, 1, ittype, ithandle, newrect);
    { récupérer les coordonnées du rectangle (newrect) }
    pensize(3, 3);
    { crayon graphique a 3 pixels }
    insetrect(newrect, - 4, - 4);
    { agrandir rectangle }
    frameroundrect(newrect, 16, 16);
    { tracer rectangle en gras autour du premier }
    pensize(3, 3);
    { retour du crayon à 1, 1 }
    penpat(white);
```

```
{ pour enlever les rectangles entourant les EditText }
{ Pas besoin de framerect, penpat(white) suffit. }
repeat
      modaldialog(nil, i_hit);
     hdl := dh(mydia, 3); { on récupère le handle }
      getitext(hdl, ggstr);
     { on récupère le texte dans ggstr }

      if (length(ggstr) = 39) then { test longueur zone }
       begin
       sysbeep(1);
       delete(ggstr, 39, 1);
       end;

      if ggstr[1] = ' ' then
{ comme le premier caractère de cette zone ne doit pas }
{ être vide, un bip d'avertissement. Ce traitement est }
{ spécifique à l'application dont nous l'avons tiré, pas au DM }
       begin
       sysbeep(1);
       delete(ggstr, 1, 1);
       end;

     { on ré-affiche le texte contenu dans ggstr }
      setitext(hdl, ggstr);

until i_hit = 1;
{ suivi du DisposDialog }
```

Dans un dialogue demandant le nom d'un fichier, on peut laisser en grisé (on le **disable** en fixant la valeur de ce *contrôle* à 255 - voir la routine HiliteControl du Control Manager) le bouton donnant accès à la suite du programme. Cliquer à l'intérieur ne donnera rien. Si c'est le premier élément (**item**) d'une liste d'items (ressources de type DITL), frapper sur le retour chariot **court-circuitera** l'item disbled. On prévoira en ce cas **deux boucles** repeat..until.

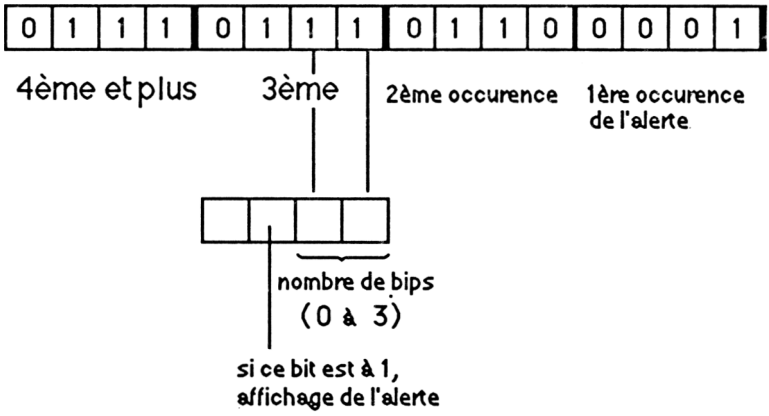
Si le fait d'appuyer sur Retour Chariot ou Entrer produit le même effet que cliquer dans le premier item du DITL, frapper sur retour chariot entrainera donc l'action que vous avez **prévue** (si vous programmez) pour le bouton du DITL listé ci-dessus. Autant l'entourer d'un rectangle en caractères gras, comme nous l'avons fait dans cet exemple.

Passons à présent à une caractéristique des alertes. Lorsqu'une alerte apparaît plusieurs fois en séquence, quelques différences peuvent se manifester d'un affichage de l'alerte à l'autre: elles sont déclenchées par le programmeur.

Chaque alerte a quatre niveaux d'affichage. Les trois premières alertes affichées peuvent avoir un "look" variable, mais pas la quatrième ni les suivantes. En examinant la ressource de type ALRT suivante, nous allons voir les variantes d'une alerte.

```
Type     ALRT (32)
     ,500
      100 70 180 330
      500
      7761
```

La dernière ligne, 7761, remplit le long mot de l'alerte:



Dans l'exemple choisi, 7761 signifie: à la première occurence de l'alerte, elle sera invisible (bit 2 = Ø), mais on entendra un bip (bit Ø = 1).

Lorsque viendra l'alerte pour la seconde fois, on la **voit** (bit 2 = 1), et on entend 2 bips (bit 1 = 1).

Le total vaut 4 (valeur du bit 2) + 2 (valeur du bit 1) = 6.

A la troisième et quatrième apparition de l'alerte (et aux apparitions ultérieures), 3 bips sont émis au lieu de 2 donc, le bit Ø = 1 pour un total de 6 + 1 = 7.

LE DIALOG MANAGER

Le DM s'occupe de gérer tous les événements qui concernent un Dialogue. Donc, pas besoin d'update event pour retracer un dialogue qui ne comprend que ses items. Ceci est valable **aussi** en ce qui concerne les PicItem. Par contre, si vous avez entouré un bouton d'un rectangle gras, à vous de prévoir la séquence d'update qui le retracera.

Vous commencez à être familier avec la séquence d'initialisation des managers:

```
InitGraf(@thePort);      { QuickDraw }
InitFonts;               { Font Manager }
InitWindows;             { Window Manager }
InitMenus;               { Menu Manager }
TEInit;                  { TextEdit, and finally: }
InitDialogs(resumeProc:ProcPtr);
```

{ ou nil, ou encore, *@aresume*, ***aresume*** est une procédure que vous écrivez, qui contient un exittoshell pour ne pas se *planter.* Sauf en cas de souris bloquée }

InitDialogs fixe un pointeur sur une procédure qui s'exécute en cas d'erreur système (qui affiche généralement une jolie alerte), installe les bips...

On peut définir un dialogue, tout comme une fenêtre, depuis le programme (en fournissant les paramètres), ou se servir de ressoureces définies dans le Type DLOG et le Type DITL (cf plus haut).

GetNewDialog(dialogId: integer; wStorage: Ptr;
behind: WindowPtr) : DialogPtr;

se sert des ressources citées, ou encore

NewDialog(wStorage: Ptr; boundsRect: Rect; title: Str255;
visible: boolean; theProc: integer;
behind: WindowPtr; goAwayFlag: boolean;
refCon: LongInt; itmLstHndl: Handle) : DialogPtr;

Si **wStorage** est mis à NIL, le dialogue est créé dans le heap.

POINTER(-1) dans le paramètre **behind** place le dialogue sur l'écran.

Si vous utilisez la procédure:

ErrorSound(soundProc: ProcPtr);

vous pouvez créer vos propres messages sonores.

Vous pouvez utiliser le même texte (static) dans deux contextes légèrement différents: ici, le premier paramètre est **"quitter"** et le second **"fermer"**.

Le texte du paramètre peut même être un nom de fichier,

contenu dans une STR255 comme dans l'exemple suivant:

```
      Type     DITL
      ,500 (32)
      2
      BtnItem Enabled
      80 10 100 50
OK

      StatText Disabled
      40 70 55 320
Le document "^Ø" a pris feu...

      IconItem Disabled
20 15 52 47
```

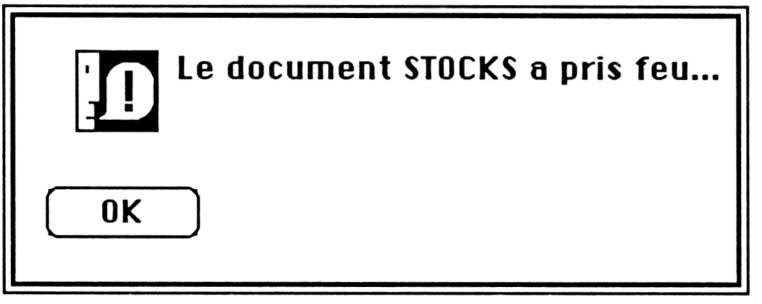
En utilisant:

ParamText(cite0,cite1,cite2,cite3: Str255);

comme suit:

ParamText(Filename,",",");

Le dialogue apparaît ainsi:



Lorsque vous en aurez fini avec un dialogue, utilisez **CloseDialog** si vous vous êtes servi du champ **wStorage**, et sinon, parce que vous aurez affiché le dialogue en utilisant GetNewDialog (ou NewDialog using NIL), ce sera **DisposeDialog** que vous appellerez.

L'exemple d'affichage de fenêtre a aussi appelé **GetdItem**, **GetlText**, et **SetlText**.

EVENEMENTS DANS LES DIALOGUES

procedure **ModalDialog**(filterProc:ProcPtr;
var itemHit: integer);

Nous avons vu l'emploi de ModalDialog dans l'exemple commenté. Tout événement généré par un utilisateur, comme le clic de la souris hors d'un bouton, la frappe de texte, est géré par le DM. Vous pouvez vouloir écrire une procédure pour gérer le Retour Chariot ou Enter intempestif. ModalDialog appelle aussi **SystemTask** (une procédure du Desk Manager qui restaure certaines zones du système. Elle devrait toujours se trouver en début d'une boucle d'événements).

La fonction **IsDialogEvent**(event: eventRecord): boolean;

est utilisée lorsqu'un dialogue comporte des EditText. Ensuite, si elle retourne TRUE, on appelle la fonction suivante:

DialogSelect(event : eventRecord
var theDialog: DialogPtr;
var itemHit : integer): boolean;

qui rend entr'autres, le numéro de l'item frappé. Et cela permet de traiter l'EditText, ou le radio-button, ou le bouton en fonction du no d'item. Dans une boucle d'événements, par exemple.

DAVANTAGE DE DM

Afin d'éditer du texte dans un dialogue a-modal, on peut utiliser les procédures suivantes:

DlgCut(dialog: DialogPtr);
DlgCopy(dialog: DialogPtr);
DlgPaste(dialog: DialogPtr);
DlgDelete(dialog: DialogPtr);

Si on veut modifier la police de caractères du dialogue, il faudra utiliser la procédure suivante

SetDAFont(fontNum: integer);

Retournons aux alertes!

Alert(alertId: integer; filterProc: ProcPtr): integer;

La seule tâche d'une alerte, c'est d'avertir l'utilisateur qu'il se passe quelque chose d'anormal ou de dangereux. Si votre alerte a une autre fonction, comme par exemple saisir du texte, c'est un *dialogue*, pas une alerte.

StopAlert, **NoteAlert** et **CautionAlert** ont le même rôle, sauf qu'elles affichent respectivement les icones de Stop, Note et Caution aux emplacements 10,20,42,52 du coin topLeft de l'alerte (la différence est de 32, taille de l'icone).

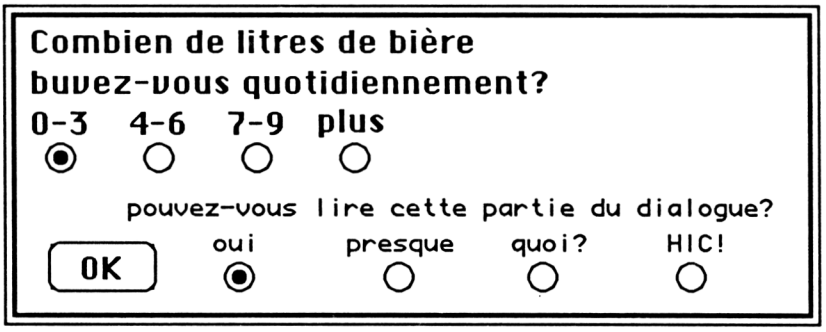
Si vous sentez qu'un utilisateur peut ôter un disque sur lequel il y a des ressources utilisées par le programme, chargez celles-ci, "gelez"-les, et rendez-les non purgeables en utilisant la procédure:

CouldDialog(DlgId: integer);

où l'entier est le no de ressource dans le type DITL. Quand vous n'en avez plus besoin,

FreeDialog(DlgId: integer);

En fin de dialogue, l'utilisateur clique dans OK ou SUITE, selon le type de texte que vous avez prévu dans le bouton.



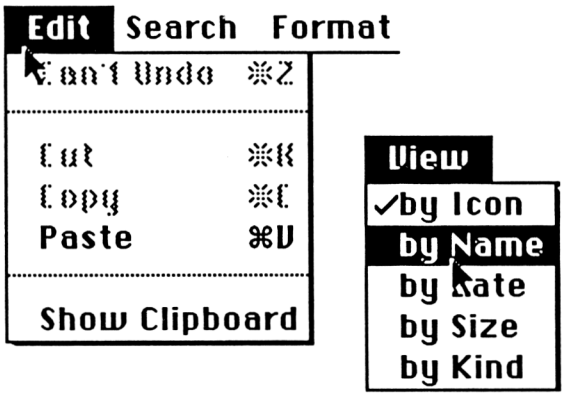
MMMMENUS

Les Menus sont une partie vitale de l'interface utilisateur du Macintosh. On peut, grâce aux Menus, savoir à n'importe quel moment quelle option nous est laissée.

Les Menus sont un bon moyen d'exploration des nouveaux logiciels.

Comme ils sont théoriquement présents sur l'écran, il n'est nul besoin de procédure complexe pour retourner aux Menus (du genre Escape...).

Un item de Menu peut être inversé ("hilited", ou highlighted en anglais) lorsque le curseur passe au-dessus. Ou encore, il peut être en grisé. Une routine du Menu Manager (MM) vous permet de dé-activer (disable) les Menus.



Si le curseur pointe vers un élément en inversé, le MM provoque le "frétillement" du texte de cet élément lorsqu'on relâche le curseur. Rien ne se passe si le curseur est relâché sur un élément dé-activé.

La barre de Menus a une hauteur de 20 pixels. Le premier des Menus, c'est le Menu Pomme. Il contient en général les renseignements sur le programme, parfois sur le programmeur. Et aussi, les DRIVERS, applications qui peuvent s'exécuter en même temps que celle que vous avez écrite, ou que vous utilisez. Alors, c'est **leur** fenêtre qui est active, recouvrant la vôtre.

Vous pouvez aussi voir figurer dans un Menu un raccourci du genre Command-Lettre (cf Fig. ci-dessus). Cette caractéristique remplace le fait de relâcher le bouton de la souris sur un item. Elle est commandée dans le corps du programme. Rien d'automatique. En quelques occasions, les raccourcis sont plus ergonomiques que la souris, c.a.d. que cela prend moins de temps et d'énergie pour arriver au même résultat.

On peut placer jusqu'à 20 items dans un menu, moins 1 pour ceux d'entre eux qui ont une icone - dans la partie de gauche. Si vous utilisez MacDraw, vous verrez certains items flanqués d'un pattern. Ici encore, vous pouvez définir les Menus dans le corps du programme, ou dans un Type MENU du fichier des Ressources.

Si vous placez des icones dans vos Menus, elles seront lues

depuis le fichier des ressources, prises dans des Type ICON.

Vous pouvez "insérer" dans un Menu des ressources d'un certain type. C'est le cas de MacWrite, avec ses "**fonts**". C'est aussi le moyen dont on se sert pour placer les ressources de type DRVR (ici, accessoires de bureau) dans le Menu pomme. Reportez-vous aux ressources de type MENU dans le Resource Manager.

LE MENU MANAGER

La première opération qui touche au Menus s'effectue en début de programme. C'est...l'initialisation du Menu Manager:

InitMenus, sans paramètre, **après** InitWindows.

Si votre Menu est défini dans le code du programme, vous utiliserez **NewMenu** et **AppendMenu**.

NewMenu(menuId: integer; menutitle: Str255): menuHandle;

crée en mémoire une structure de Menu avec les paramètres fournis depuis le programme, et retourne un menuhandle sur cette structure.

AppendMenu(menu: menuHandle; data: Str255);

se sert du MenuHandle afin d'ajouter l'item - défini par son nom, une chaîne de caractères - en queue (Append) du Menu correspondant.

AppendMenu(MyMHndl,'Cut;Copy;Paste');

Si vous définissez les Menus dans le fichier des ressources, c'est **GetMenu**, et **InsertMenus** que vous utiliserez.

function **GetMenu**(rsrcId: integer): menuHandle;

GetNewMenu identifie le Menu par son numéro dans le type MENU, et retourne un menuhandle. Si vous voulez appeler GetNewMenu plusieurs fois, appelez **ReleaseResource** pour désallouer le/les menuHandle.

Lorsque vous aurez obtenu un MenuHandle pour **chaque** composant (titre et items à l'intérieur) **du Menu**, et l'aurez éventuellement placé dans un array de of menuHandle, utilisez:

InsertMenus(menu:MenuHandle; beforeId: integer);

pour chaque élément du Menu; ensuite:

DrawMenuBar fera apparaître la barre de Menus à l'écran.

A présent vient le temps d'utiliser les Menus. La fonction:

MenuSelect(startPtr: Point): Longint;

utilisée après un **FindWindow** (cf. chapitre précédent), retourne un longint composé de deux parties bien distinctes: l'entier de gauche, et l'entier de droite. Le **Hiword**(MyLongInt), entier de gauche, indique dans quel Menu la souris a été relâchée (c.a.d. Pomme, Fichier,...). On peut définir des constantes dans le programme, comme AppleMenu=1, FileMenu=2, etc. Pourquoi le mouse up n'est-il pas testé? MenuSelect le fait à votre place!

L'octet de droite vous rend le quantième de l'item sur lequel a été relâché le bouton de la souris.

Avec ces deux coordonnées, il vous est possible de prendre une action. En général, par un case..of.

Par exemple, si vous cliquez dans le Menu "**Fichier**" et relâchez le bouton sur "**Imprimer**", l'entier de gauche contiendrait **2**, et celui de droite, **7**.

A présent, **Hilite**(Ø) enlève l'inversion couleur de la barre de Menu, et tout ce petit monde se retrouve dans le premier programme commenté.

Rien ne se passe lorsqu'on relâche la souris sur Cut, Copy ou Paste. Ce programme sert simplement de démonstration à l'utilisation des Menus et ressources et de la boucle d'événements, ainsi que des raccourcis.

Une icone est aussi incluse. Et l'item **Quitter** est en relief.

Lorsqu'on relâche le bouton de la souris dans le Menu Pomme, sur l'item "A propos de YANN", apparaît un dialogue. Il disparaît dès qu'on appuie sur YEAH. Ce programme appelle aussi les accessoires de bureau.

Nous avons été obligé d'inclure la boucle d'événements: c'est d'abord nécessaire, car sans elle, on ne peut entrer dans les Menus. Ensuite, c'est un moyen de la voir vivante. Sans boucle d'événements, vous verriez les Menus durant 1/4 de seconde, et pfuitt, disparus, fin de programme.

Cet exemple n'est pas compliqué. Notre seul but en l'écrivant était de vous voir sourire, en considérant, soulagé, à quel point tout cela est simple.

MonProg

```
PROGRAM YANN1;
{ Dessine la barre de Menus et ouvre les Menus }

{ *** ici, en LisaPascal. D'autres exemples seront  *** }
{ *** fournis en TML Pascal  *** }
{$X-}
{$U-}

USES {$U Obj/MemTypes } MemTypes,
     {$U Obj/QuickDraw} QuickDraw,
     {$U Obj/OSIntf  } OSIntf,
     {$U Obj/ToolIntf } ToolIntf;

CONST

  appleID  = 128; { no d'IDs du menu Pomme, }
```

```
  fileID     = 129; { Fichier,}
  editID     = 130; { Edit }

  appleM  =    1;    { index pour menuhandle }
                     { dans la table mymenus }
  fileM      =    2;
  editM      =    3;
  MenuBase=   127;{ constante pour ramener les }
                     { nos d'ID aux index }
  LastMenu=    3;

VAR

  MyMenus: ARRAY[1..LastMenu] OF MenuHandle;
       { table de 3 handles vers les menus }

  doneFlag:     BOOLEAN; { on le met à TRUE }
  { dès qu'on a repéré que l'utilisateur a cliqué sur  "Quitter" }
  myEvent :     EventRecord;
  myWindow:   WindowPtr;
  whichWindow:WindowPtr;  { pointeur sur la fenêtre où }
                          { l'utilisateur a cliqué }

PROCEDURE SetUpMenus;     { Installe les menus }
  { et dessine la barre de menus }

 VAR i: INTEGER;

 BEGIN
  { Lit les menus dans le fichier des ressources, et stocke }
  { les menuHandles dans la table  myMenus }

 FOR i:=1 TO LastMenu DO
  BEGIN
   MyMenus[i]:=GetMenu(i+MenuBase); { 127 + i }
   InsertMenu(myMenus[i],0); {installe chaque menu }
      { dans la barre de menus }

   IF i = 1 THEN
      AddResMenu(myMenus[1],'DRVR');
      { ajoute les accessoires de bureau au menu pomme }
  END;    { FOR }

 DrawMenuBar; {  affichage de la barre de menus }
```

```
 END;       { SetUpMenus }

{$S Dialog}
{-------------------------------------------------------------------------------}
PROCEDURE DrwDialog;
   { dessine le dialogue }

VAR    itemHit, Temp   : INTEGER;
         myDialog          : DialogPtr;
BEGIN
   myDialog := GetNewDialog (257, NIL, POINTER (-1));
   { déjà vu }

   REPEAT
         ModalDialog (NIL, itemHit); { gestion des événements }
   UNTIL itemHit = 1;                  { le bouton est l'item no 1 }
   { on sort de la boucle }
   DisposDialog (myDialog);
   { on se débarasse du dialogue }
END; {* Drwdialog *}

{ ***************** }

PROCEDURE ExecCommand (theResult: LONGINT);

{ Exécute l'item de menu retourné par theResult, }
{ retourné par MenuSelect }

  VAR  theItem: INTEGER;       { no d'item de menu }
    theMenu: INTEGER; { no de menu }
    name: Str255;     { nom d'accessoire de bureau }
    temp: INTEGER;

  BEGIN
  theItem := LoWord(theResult);
  theMenu := HiWord(theResult);


  CASE theMenu OF  { menu ID}

    appleID: { si on se trouve dans le menu pomme }
    { on affiche le dialogue "A Propos de..." }
          BEGIN
           IF theItem = 1 THEN
```

```
        DrwDialog
    ELSE
     BEGIN
       GetItem(myMenus[1],theItem,name);
  { appelle le nom d'accessoire de bureau }
      temp := OpenDeskAcc(name);
  { et ouvre celui-ci. Le champ temp n'est pas utilisé }
     END; { ELSE }
 END; { appleID}

fileID:
  doneFlag := TRUE; { lorsque l'utilisateur a cliqué sur }
  { Quitter dans le menu Fichier, on positionne un booléen }
  { à TRUE }

editID:
  BEGIN    { on ne fait rien ici - du moins, dans cet exemple }
  END; { editID}

 END;  { menu case}
 HiliteMenu(0); { MenuSelect affiche le Menu sélectionné }
          { en inverse. Ici, on restaure l'affichage normal }
 END;  { ExecCommand}

{$S }

BEGIN { programme principal }

moremasters;
moremasters;
unloadseg(@moremasters);
{ pas utile ici, mais bonne habitude }

{ Initialisations: }
InitGraf(@thePort);   { initialise QuickDraw }
InitFonts;            { initialise Font Manager }
FlushEvents(everyEvent,0); { calls OS Event Manager to flush }
                      { any previous events }
InitWindows;          { initialise Window Manager }
InitMenus;            { initialise Menu Manager }
TEInit;               { initialise TextEdit }
InitDialogs(NIL);     { initialise Dialog Manager }
InitCursor;           { appel à QuickDraw afin de }
                      { mettre le curseur 'flèche' sur l'écran }
```

```
.....
{ dans une application normale, on placerait ici une }
{ portion de code gérant la mémoire }
.....

SetUpMenus;          { Affichage de la barre de menus }
UnloadSeg(@FixMenus); { enlevez-moi ce segment }

doneFlag := FALSE; { FALSE: deviendra TRUE lorsqu'on }
                   { passera  dans ExecCommand, si l'utilisateur }
                   { clique sur "Quitter". On pourrait s'arrêter ici, mais }
                   { ce serait la fin de ce programme. On écrit donc }
          { une boucle d'événements, qui gère les événements}
          { de tous types }

{ Boucle de logique des événements }


REPEAT
  SystemTask;          { restauration de zones par le }
                       { Desk Manager. }

  IF GetNextEvent(everyEvent,myEvent)
    { appel au Toolbox Ev.Mgr pour placer dans }
    { l' event record myEvent  l'événement que l'application }
    { doit gérer. Nous avons ici sélectionné tous les }
    { événements en fournissant everyevent  comme masque }

    { comme il y a, en gestion, beaucoup de dialogues, }
    { nous plaçons ici IsDialogEvent suivi de DialogSelect }
    { nous récupérons ainsi l'item de dialogue frappé }
    { et la fenêtre active. Vous êtes libre de faire autrement. }
    { Tout dépend de l'application }

     CASE myEvent.what OF          { selon le type d'événement }

         mouseDown: { bouton de la souris appuyé }

         CASE FindWindow(myEvent.where,whichWindow) OF

    { appel au Window Manager pour savoir où l'utilisateur }
    { a cliqué dans la fenêtre }

              inSysWindow:
```

```
        { dans un accessoire de bureau. Appel au Desk Mgr }
        {  qui le gère }
        SystemClick(myEvent,whichWindow);

            inMenuBar:          { dans la barre de menus }
                ExecCommand(MenuSelect(myEvent.where));
    { voyez ce que fait la procédure ExecCommand }

            inDrag:             { dans la barre de titre }
                BEGIN
    { ici, on ne fait rien. Ce n'est qu'un premier programme. }
    { en général, on appelle DragWindow... }
                END;        { inDrag }

{ *** }
            inContent:      { corps de la fenêtre d'application: }
    { la ou les actions à exécuter dépendent de la spécificité de }
    { l'application }

                BEGIN
                END;        { inContent}

{ *** }
            inGoAway:       { dans la boîte de fermeture }
            { (GoAway) box. En général, if trackGoAway... }
            { suivi d'une sauvegarde éventuelle, puis, }
            { fermeture de la fenêtre }
                BEGIN
                END;        { inGoAway}
{ *** }
            inGrow:         { dans la boîte de dimensionnement }
                BEGIN
                END;        {inGrow}
{ *** }
        END;        {mouseDown}

        keyDown, autoKey: { événements de type clavier }
            BEGIN
            END;        { keydown, autokey }
{ *** }
{ *** }
        activateEvt:
            BEGIN
            END;        { activateEvt }
```

```
{ *** }
{ *** }
        updateEvt:    { mise à jour du contenu d'une fenêtre }
            BEGIN
            END;      { updateEvt }

    END;      { event case}

UNTIL doneFlag;       { end repeat...until }

END.
```

Vous pourrez voir les écrans produits par ce programme (obtenus en appuyant sur "Verrouillage Maj.", Commande - 3, donnant ses Screen Ø, Screen 1,...) immédiatement après le listage du fichier des ressources. D'abord, vous avez pu noter à quel point il est facile de produire des menus et une boîte de dialogue.

On n'a utilisé ici la boucle d'événements qu'à fin de gérer les menus. Vous avez pu constater que beaucoup des CASE ont été laissés vides.

Voici donc le fichier des ressources:

```
*  yann -- Draws Menu Bar
*  Ecrit par YANN FAIN

yann.Rsrc

* nom du fichier de sortie du RMaker

Type MENU
  ,128
*** menu Pomme (ASCII = 20 ) ***
*** dans une fonction NewMenu, on écrirait: MyTiltle[1]: = 20; ***
  \14
    About Downhill Greg...
   (-

  ,129
  YourFile
    Quit^2<O/Q
* icone + outline + raccourci

  ,130
  MyEdit
    (Undo/Z
    (-
    Cut/X
    Copy/C
    Paste/V
    Clear

Type ICON
     ,257 (32)
*** du dialogue. Le DITL qui suit appelle cette ressource ***
     FFFF FFFF
     A000 0001
```

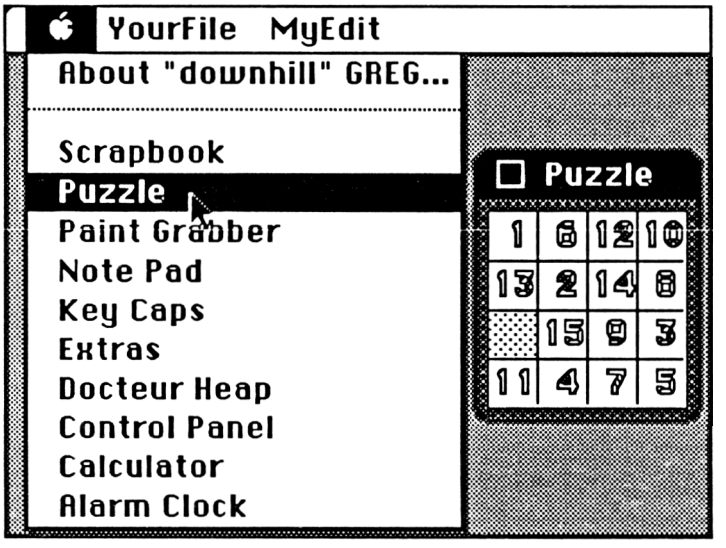
```
*  . . . . . 32 lignes de huit chiffres hexadécimaux.
    00FF FE00
    ,258 (32)

*** dans l'option "Quit" ***
    0000 0000
    0000 0000
   . . . . . .
    03FE 0FF0
    0000 0000
    0000 0000
    0000 0000
    0000 0000

Type DLOG
*  Dialogue "A propos de Ceprog", en général le premier du
Menu Pomme.

    ,257 (32)
*  DLOG#257, purgeable.
80 50 185 420
*  coordonnées des coins (top left bottom right)
Visible 1 NoGoAway 0
* syntaxe du RMaker du Workshop.
*  Visible, no GoAway box, proc. par défaut
257
* appellera le DITL# 257 (qui suit)

Type DITL
    ,257 (32)
    5
*  5 items
BtnItem Enabled
*  Bouton. On peut l'activer et le dé-activer.
* S'il est actif, l'utilisateur peut cliquer dedans, et le programme
* pourra tester le mousedown dans ce bouton.
71 335 80 400
YEAH
*  il contient: "YEAH"!

IconItem Disabled
10 10 42 42
257
*  l'icone, porte le numéro 257, est située en 10, 10, 42, 42

StatText Disabled
18 60 59 260
```

Premier programme court Affiche dans un dialogue ce texte et une icone
* c'était une partie du texte contenu dans cet item

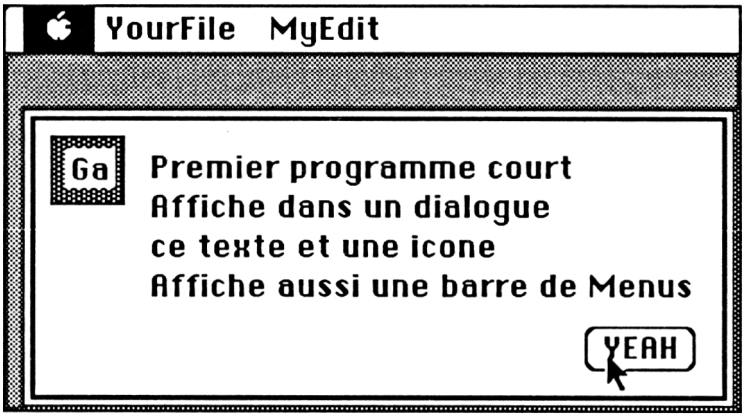
StatText Disabled
61 60 74 240
Affiche aussi une barre de menus
/* cet autre StatText contient le reste du texte

Type CODE
yann,0
* le code est en lui-même une ressource, de numéro Ø

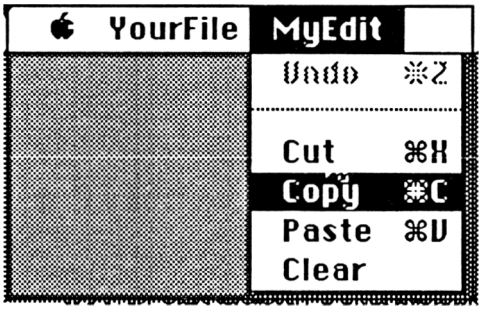
Voici ensuite les résultats: Nous pouvons appeler les accessoires de bureau. C'est le Desk Manager qui se charge d'ouvrir et fermer les accessoires de bureau.



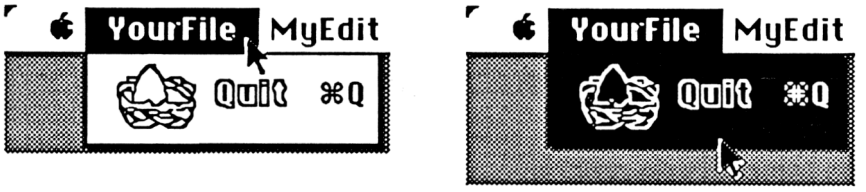
Regardez en page suivante le dialogue qui apparaît sur la fenêtre. Il pousse un bip quand on clique en dehors du bouton YEAH. Nous l'avons programmé pour disparaître dès que l'on clique sur ce bouton.



Nous avons écrit les Menus en anglais pour bien montrer que l'on peut écrire le texte que l'on désire, et que rien n'est lié à la version du système, sauf (mais on ne peut le savoir par cet exemple) les assignations de touches. Cliquer sur ces touches ne produira rien, puisque nous avons laissé leur programmation vide.



La boucle d'événements utilisée ici est de type standard. Vous pouvez créer des boucles d'événements plus sophistiquées, et même, avec des tables d'événements. Ce qui est important est que "ça marche".



Vous voyez l'icone, le "Quit" en outline et le raccourci Commande-Q. Reportez-vous au DITL correspondant pour savoir comment on organise tout cela depuis le fichier des ressources. La programmation des raccourcis s'effectue depuis le programme. C'est pourquoi dans un programme que vous voulez modifier avec l'utilitaire ResEdit afin d'y rajouter des raccourcis, vous devrez regarder dans le programme s'il y en a déjà un. C'est le signe que la programmation de cette séquence a été prévue.

Les options offertes dans la barre de menu ont été prévues dans le prpgrammation, mais les blocs de code sont vides. C'était donc un premier programme, simple.

DAVANTAGE SUR LE MENU MANAGER

Nous venons d'évoquer l'option "Quit" du Menu, et ses variantes d'affichage par rapport à un menu classique. Voici à présent toutes les possibilités d'affichage de menus.

La chaîne de caractères (Str255) fournie par le fichier des ressources, ou par le champ **data** (voir **"NewMenu"**), est faite de texte, mais peut aussi comporter d'autres caractères:

; ou RETURN	:	sépare les options; voir AppendMenu;
^n	:	une icone est placée à gauche du texte lors de l'affichage. *n* est compris entre 1 et 9, et le Menu Manager ajoute 256 à ce nombre. Ce sera le numéro de l'icone dans son type ICON.
!	:	caractère spécial, comme un check mark;
<	:	changement de style (B,I,U,O,S,...);
	:	Bold, Italic, Underlined, Outline, Shadow,...
/	:	raccourci (équivalence Commande-Lettre)

(: apparaîtra en grisé lors du premier affichage. Pourra éventuellement être activé par l'ordre EnableItem

On peut modifier le style et récupérer le style actuel d'une icone (en cours de programme), pour le modifier le cas échéant.

La fonction **ClearMenuBar**:handle; ne libère pas la place reservée par le handle. On peut l'utiliser avant d'afficher un nouveau menu, après que le premier ait été affiché.

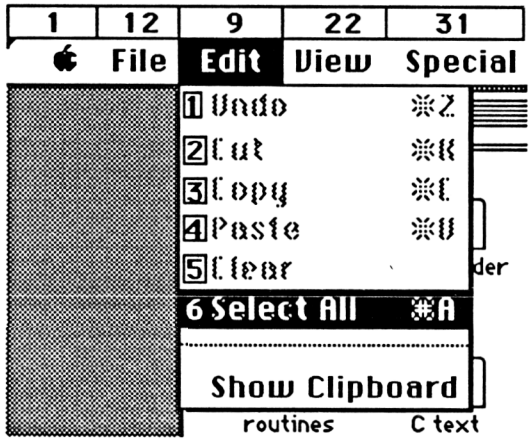
Nous avons vu dans le programme l'utilisation de **SetMenuBar**(menuBar: handle); ensuite, on appelle DrawMenuBar.

Aussi importante que MenuSelect, la fonction:

MenuKey(ch: char): LongInt;

sera utile si vous incluez des raccourcis. Voici le contenu du Longint qu'il retourne si vous écrivez:

MenuSelect(thePoint), ou MenuKey('A'):



high word: 9 | low word: 6

Les ID de Menu ne sont pas nécessairement ascendantes (comme dans l'exemple ci-dessus).

Voici pratiquement le même exemple que celui fourni en Pascal, mais écrit en "C" (Consulair) cette fois.

```
/* Programme de démo de boucle, écrit par Yann Fain. */

  #define UseWithStdLib

/* Déclarations */

#include "MacCDefs.h"          /* librairies */
#ifdef UseWithStdLib           /* librairies conditionnelles*/

#include "Stdio.h"
#else
   extern struct P_Str *CtoPstr();
   int strlen(str) char *str;                    /* pour pointer sur le
                                                    caractère de la str */
     {int i=0; while (str[i++]); return i-1;}

  #endif                                         /* end condition */


  #include "Window.h"
  #include "Events.h"
  #include "TextEdit.h"
  #include "Menu.h"

    /* Déclarées dans le programme */

MenuHandle DeskMenu;
MenuHandle FileMenu;
MenuHandle EditMenu;

#define Desk_ID 200
#define File_ID 201
#define Edit_ID 202

#define False 0
#define True 0xFF /* 255 */

  /* Déclarées ailleurs */

#ifdef UseWithStdLib
   extern WindowPtr console; /* fen. Lib TTY standard */
#endif
/* Code */
```

```
Init()
{

InitMenus();
      /* utilise NewMenu et AppendMenu, au lieu du  */
      /* GetMenu dont nous nous sommes servis en Pascal */
      /* FixMenus: procédure équivalente du prog. Pascal */
      /* Desk Accessory menu */
DeskMenu = NewMenu(Desk_ID,CtoPstr("\024"));
AppendMenu(DeskMenu,
CtoPstr("A Propos de Ceprog"));

 AddResMenu(DeskMenu, 'DRVR');
/* rajoute les accessoires de bureau */
 InsertMenu(DeskMenu, 0);
      /* Menu "File" */
 FileMenu= NewMenu(File_ID, CtoPstr("MyFile"));
 AppendMenu(FileMenu,
CtoPstr("Quit<O/Q"));
/* ici, les options de raccourci sont rajoutées dans le corps du
      programme */

InsertMenu(FileMenu, 0);

      /* Edit menu */
EditMenu = NewMenu(Edit_ID, CtoPstr("YourEdit"));
AppendMenu(EditMenu,
CtoPstr("Undo;(-;Cut/X;Copy/C;Paste/V;Clear"));
InsertMenu(EditMenu, 0);
DrawMenuBar();   /* Déjà vu? */
}

/* PROGRAMME */
main()                  /* phase d'appel */
{
char c, *str;
short windowcode;
long menuresult;
/* récupèrera les nos de menu et d'item de menu */
EventRecord event;
WindowPtr mouseWindow;
if (CatchSignal())  ExitToShell();
/* on sort dès que "Signal" est détecté */
Init();
InitCursor();
FlushEvents(0xFFFF);
```

```
/* boucle d'événements */

 while (True) /* boucle jusqu'à CatchSignal */
 {
 SystemTask();     /* pour les accessoires de bureau */
 /* on devrait aussi placer ici un TEIdle s'il y a du texte */

 if (GetNextEvent(everyEvent, &event)) /* syntaxe différente */
/* de celle du Pascal */
{
switch ( event.what )
 {
 case mouseDown:
  {
      windowcode = FindWindow(&event.where, &mouseWindow);
    /* ici encore, la syntaxe est différente */

   switch ( windowcode )
        {
        case inContent:       /* vide, ici */
         {
         break;
         }
        case inDrag:
         {
         break;
         }
        case inGrow:
         {
         break;
         }

        case inGoAway:
         {
         break;
         }
/* nous n'avons pas répété les explications concernant ces
entiers rendus par FindWindow */

        case inMenuBar:
         {
          DoMenu(MenuSelect(&event.where));
  /* équivalent dans le programme Pascal: ExecCommand */
          break;
         }
        case inSysWindow:
         {
```

```
                SystemClick(&event, mouseWindow);
                break;
               }
             } /* end switch windowcode */
            break;

          } /* end mousedown */

        case updateEvt:
         {
         break;
         }
        case activateEvt:
         {
         break;
         }
        case autoKey:
        case keyDown:
         {
         break;
         }
       } /* end switch(event.what) */

     }
    }
}
/* fin de la boucle principale (d'événements) */
/* à présent suit le code des procédures et fonction appelées */

/* DoMenu récupère dans menuresult le no de menu, et le no
d'item de menu sur lequel on a relâché la souris */

 DoMenu(menuresult)
   long menuresult;
   {
   short menuID, itemNumber;

   menuID = HiWord(menuresult);
   itemNumber = menuresult;
/* le passage du long menuresult dans itemnumber (short)
tronque la zone menuresult sur la gauche */

   switch ( menuID )
    {
     case Edit_ID:
    {
    switch ( itemNumber )
```

```
    {
    case 1: break;          /* item 1; Undo, etc. */
    case 2: break;          /* item 2 */
    case 3: break;          /* item 3 */
    case 4: break;          /* item 4 */
    case 5: break;          /* item 5 */
    case 6: break;          /* item 6 */
    }
    break;
  }
   case Desk_ID:
  {
  struct P_Str AccessoryName;
  GetItem(DeskMenu, itemNumber, &AccessoryName);
  OpenDeskAcc(&AccessoryName);
  DrawMenuBar();
  break;
  }
   case File_ID:
  {
   Signal("End"); /* on termine ainsi */
/* ensuite, on se sert de CatchSignal */
  break;
   }
  }
  HiliteMenu(0);
 }
```

Ce programme requiert beaucoup moins d'explications que le précédent. La syntaxe utilisée, légèrement différente de celle du Pascal, est encore familière aux "Pascalistes".

'Si j'utilise le "C", puis-je créer mes propres ressources et utiliser les routines écrites dans l'exemple en Pascal?' La réponse est...oui. Mais il faudrait remplacer le code précédent d'*InitMenus* par le suivant:

```
/* Code */
 Init()
 {
   InitMenus();
    {
        MyMenu1 = GetMenu(128);
        InsertMenu(MyMenu1,0);
        AddResMenu(MyMenu1,'DRVR');
         MyMenu2 = GetMenu(129);
        InsertMenu(MyMenu2,0);
```

```
        MyMenu3 = GetMenu(130);
        InsertMenu(MyMenu3,0);
        DrawMenuBar();
    }
```

En plaçant les ressources (dans ce programme, uniquement de type MENU) dans un fichier des ressources nommé GacDemo1.r, on obtiendrait, une fois passé à la moulinette du RMaker un GacDemo1.Rel.

```
gacdemo1.Rel
* nom du ".rel" résultant

Type MENU
  ,128
*** la pomme ***
  \14
    A Propos de Ceprog...
   (-

  ,129
  YourFile
    Quit<O/Q

  ,130
  MyEdit
    (Undo/Z
    (-
    Cut/X
    Copy/C
    Paste/V
    Clear
```

On exécute l'ensemble par une phase **".job"**:

```
C       GacDemo.C       Exec    Edit
Link    GacDemo.link    Exec    Edit
```

Au moment du link, les ressources sont ainsi introduites:

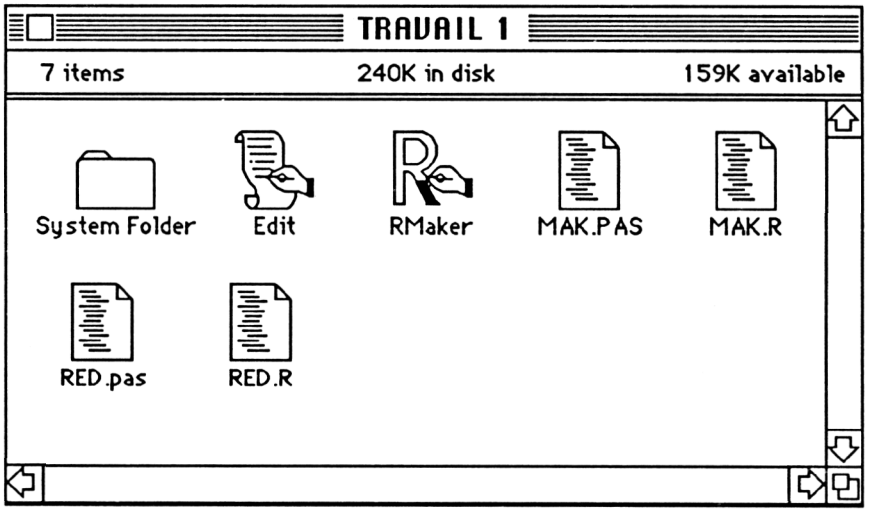
```
!start
Stdlib
MacCSioLib
MacCTTY
MacCmem
MacCstrings
MacCstringutil
MacCutil
MacCio
MacCiosupp
MacCioprim
MacCFileio

GacDemo

/resources Gacdemo1.rel
$
```

On peut aussi se servir du Resource Editor (REdit ou ResEdit) pour "Paster" les ressources au code obtenu (.rel aussi) en sortie de link. Mais alors, pas de /resources Gacdemo1.rel dans le fichier du link.

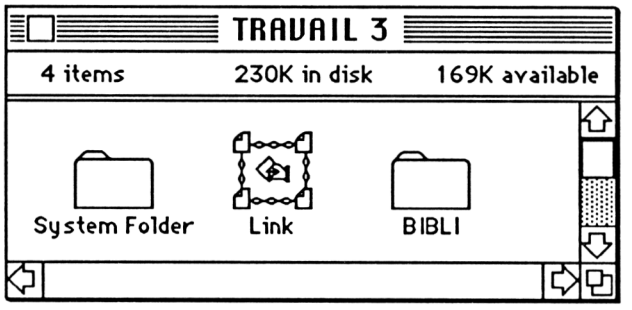
L'exemple écrit en Pascal fournissait un affichage de fenêtre que vous n'avez pas trouvé en "C". Ces exemples étaient là surtout pour les menus.

Voici la manière de travailler quand on débute en TML.

A partir des deux disquettes reçues de chez TML, ou de chez l'importateur, on crée une première disquette contenant l'Editeur et RMaker. Après avoir écrit un source des ressources (ici, MAK.R) conformément à la syntaxe du RMaker de MDS, on le passe au RMaker, et on place le résultat (MAK/Rsrc.REL) sur la disquette de travail no **3**.



La Fig. ci-dessus montre la fenêtre de la disquette de travail no 1.



Quant à la disquette de travail no 3, elle contient aussi le Linker et les bibliothèques **.REL** qu'il utilise pour linker (rassembler les différents modules en une application).

La disquette de travail no **2** contiendra le compilateur, et le fichier MAK.PAS issu de la disquette de travail no **1**, après qu'on ait écrit le programme source **.PAS**; la disquette de travail no **2** contient aussi les librairies **.IPAS** les plus courantes.

La compilation produit deux fichiers (plus, lorsqu'on utilise la segmentation): un fichier MAK.**link**, et un fichier MAK.**rel.** On les

passe tous les deux depuis la disquette de travail no **2** sur la disquette de travail no **3**.

Quand tout le monde se retrouve sur cette disquette, on lance le Linker, et on sélectionne le fichier **.link** créé par le compilateur et on obtient une application.

Quand on a un peu plus d'expérience, on module selon la configuration (Mac, Mac +, RamDisk, lecteur...).

Une caractéristique intéressante du LisaPascal, c'est la possibilité de définir un type, et ensuite une constante égale à la taille du type défini:

```
type machin = packed record
        champ1: longint;
        .....
        end;
const longmachin = sizeof(machin);
```

Ceci n'existe pas en Pascal TML, qui n'accepte pas grand chose de peu conformiste, ni les instructions cycle, exit,...

Afin de savoir quelles sont les longueurs des booléens, ou d'autres types de zones, vous pourrez utiliser le segment de code suivant. De plus, il montre un exemple de blockmove (instruction fort intéressante, parfois préférable à un copybits), et d'utilisation des routines hautes du File Manager.

```
program pr1;

{$i MemTypes.ipas  }
{$i QuickDraw.ipas }
{$i OSIntf.ipas    }
{$i ToolIntf.ipas  }
{$i PackIntf.ipas  }

 const
 malong=512; { on écrit sur une grande longueur }

 type
 monrec=packed record
 lechamp:packed array[1..512] of char;
 end;

 var
 table2: packed array[0..1] of boolean;
```

```
table8: packed array[0..7] of boolean;
table16: packed array[0..15] of boolean;
table32: packed array[0..31] of boolean;
table64: packed array[0..63] of boolean;

a, b, vol1num, n_sys, io_err:integer;
long, thesp, freesp, ali: longint;
stptr: ^longint;
Mystr, v1nm: str255;
bufsys:monrec;
hdl: handle;

begin
 moremasters;
 moremasters;
 unloadseg(@moremasters); { un seul segment, donc }
 { pas besoin. Mais bonne habitude }

 initgraf(@theport);
 initfonts;
 initwindows;
 initmenus;
 teinit;
 initdialogs(nil);
 initcursor;
 { fin des initialisations }

 thesp:= ord4(1024) * ord4(2048);
 freesp:= compactmem(maxsize);
 freesp:= maxmem(thesp);
 stptr:= pointer($110);
 ali:= stptr^ - ord(18772);
 setapplimit(ptr(ali));
 thesp:= ord4(1024) * ord4(2048);
 hdl := newhandle(thesp);
 disposhandle(hdl);
 for a:=0 to 1 do
       table2[a]:=true;
 for a:=0 to 7 do
       table8[a]:=true;
 for a:=0 to 15 do
       table16[a]:=true;
 for a:=0 to 31 do
       table32[a]:=true;
 for a:=0 to 63 do
       table64[a]:=true;
```

```
   io_err := getvinfo(-1, @v1nm, vol1num, freesp);
   { -1 : no de drive du startUp disk }

   io_err := create('SYS000', vol1num, 'YANN', 'DATA');
   io_err := fsopen('SYS000', vol1num, n_sys);

   for a := 1 to 512 do bufsys.lechamp[a] := ' ';
   { mise à blanc du champ }

   io_err := setfpos(n_sys, 1, 0);
   long := malong;
   for a:=1 to 5 do
     io_err := fswrite(n_sys, long, @bufsys);
   io_err := fsclose(n_sys);
   io_err := flushvol(nil, vol1num);

   io_err := fsopen('SYS000', vol1num, n_sys);
   io_err := setfpos(n_sys, 1, 0);

blockmove(pointer(ord4(@table2)),pointer(ord4(@bufsys)),sizeof(table2));
   io_err := fswrite(n_sys, long, @bufsys);

blockmove(pointer(ord4(@table8)),pointer(ord4(@bufsys)),sizeof(table8));
   io_err := fswrite(n_sys, long, @bufsys);

blockmove(pointer(ord4(@table16)),pointer(ord4(@bufsys)),sizeof(table16));
   io_err := fswrite(n_sys, long, @bufsys);

blockmove(pointer(ord4(@table32)),pointer(ord4(@bufsys)),sizeof(table32));
   io_err := fswrite(n_sys, long, @bufsys);

blockmove(pointer(ord4(@table64)),pointer(ord4(@bufsys)),sizeof(table64));

   io_err := fswrite(n_sys, long, @bufsys);
   io_err := fsclose(n_sys);
   io_err := flushvol(nil, vol1num);
 end.
```

Ensuite, vous prendrez l'utilitaire FEdit et comprendrez, d'après l'examen en hexadécimal de votre fichier SYSØØØ, comment le Pascal TML gère les booléens. Et quelles longueur il leur attribue.

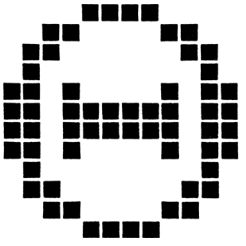
Les polices de caractères et TextEdit sont nos dernières barrières avant les exemples intéressants. A présent, aimeriez-vous vos programes assembleurs personnalisés comme suit?

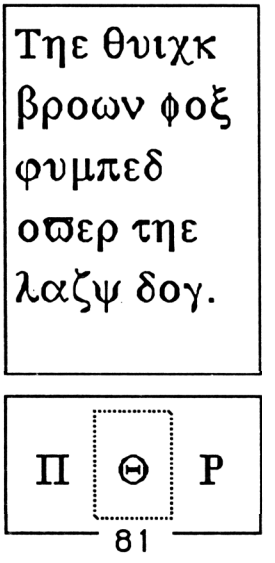
MOVE.L A6,-4(A6)
DC.W $A930
CLR.L -(SP)
MOVE #$C8,-(SP)
LEA 'at',A0
MOVE.L A0,D0
JSR CtoPstr
ΔX.Ω ∃A931

MOVE.L A0,DeskMenu(A5)

CHAPITRE XI

TEXTEDIT, FONT MANAGER





Le croiriez-vous? Ce qui est écrit dans le rectangle du haut, à droite, c'est: "The quick brown fox jumped over the lazy dog"!

Ce chapitre est dédié aux éléments qui nous manquent pour pouvoir comprendre les programmes commentés.

TextEdit est assez simple, donc, nous l'attaquons vite vite.

En utilisant MacWrite, vous avez pu constater la présence d'une barre verticale lorsque vous cliquez entre deux lettres: c'est le point d'insertion, qui vous montre la position du texte à insérer.

L'ensemble de votre texte est centré, aligné à droite, à gauche ou des deux côtés selon les règles que vous avez

sélectionnées. Cet alignement s'appelle la *justification.*

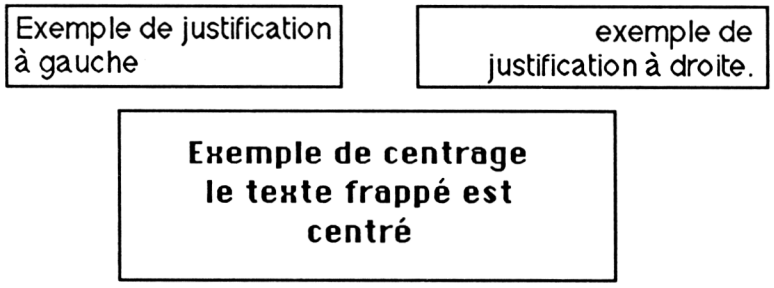
Il y a diverses manières de sélectionner du texte: double clic sur un mot, Majuscule et clic, dragging (en positionnant le point d'insertion et en descendant ou remontant la souris, bouton maintenu appuyé). Si une portion de texte est sélectionnée (le texte apparaît en blanc sur fond noir), le fait de frapper n'importe quelle lettre, ou même la touche TAB **remplace** le texte sélectionné. Suit un exemple visuel d'insertion et de substitution de texte:

Lorsqu'on traverse une zone contenant du texte, vous avez pu voir se modifier la forme du curseur (I-beam).

La position du texte est liée au grafport par la position du crayon graphique.

JUSTIFICATION

TextEdit a un jeu de routines qui permettent l'alignement à droite, à gauche, le centrage. Mais pas la "justification" des deux côtés. Dans MacWrite, cette option est *programmée.*



La Toolbox fournit des constantes pour fixer la justification. Si vous devez programmer une application, servez vous des constantes fournies. Si les valeurs changent dans une version future, vous n'aurez pas à recompiler.

```
const
    teJustLeft   =  Ø;
    teJustCenter =  1;
    teJustRight  =  -1;
```

LIMITES DE TEXTEDIT

Donc, TextEdit ne pourra aligner le texte des deux côtés. Mais il ne pourra non plus s'occuper des espaces proportionnels (rappelez-vous le champ spExtra dans le GrafPort). TextEdit ne gèrera pas non plus les tabulations. Donc, si vous voulez écrire un traitement de texte, toutes ces options sont à programmer.

TextEdit ne permet pas l'utilisation de plus d'une police de caractères ou de changement de style dans un editrecord. C'est donc à vous de gérer tout cela.

Par contre, TextEdit (appelons-le TE, maintenant qu'il nous est un peu plus familier) permet de sélectionner le texte par double-clic, dragging (cf les explications fournies plus haut), inverse automatiquement l'affichage du texte sélectionné, fournit automatiquement le point d'insertion. TE empêche aussi qu'un mot soit coupé en fin de ligne, et naturellement, les fonctions "Couper", "Copier" et "Coller. En ce cas, le texte est mis dans une zone de mémoire (scrap) propre à TE - scrap local -, et non à l'application ('desk scrap', utilisé pour passer des données d'une application à une autre).

EDIT RECORD

Le croiriez-vous? Il existe un record contenant les informations relatives au texte. Elle est contenue dans un TERec sur lequel pointe un TEPtr, auquel est relié un TEHandle.

```
TEPtr       = ^TERec;     { Pointeur sur TERec }
TEHandle    = ^TEPtr;     { Handle sur TERec }
CharsHandle = ^CharsPtr;  { Handle sur CharsPtr }
CharsPtr    = ^Chars;
Chars       = packed array[0..32000] of char;
```

L'édition de texte requiert deux rectangles. L'un d'entre eux (viewRect), est le rectangle dans lequel le texte est visible. Vous lui assignerez les coordonnées que vous voulez. Ce viewrect est contenu dans un destrect, rectangle d'entrée de texte. On peut entrer du texte en page 1 alors qu'est affichée la page 2 à l'écran.

```
TERec = record
      destRect: Rect;          { rectangle de destination }
      viewRect: Rect;          { rectangle d'affichage }
      selRect:  Rect;          { rectangle de sélection }
      lineHeight: integer;{ hauteur du texte }
                               {utile pour l'espacement vertical }
      fontAscent: integer;     { hauteur de la police de car. }
                               { depuis la ligne de base }
      selPoint:  Point;        { point de sél. de texte }
                               (mouseLoc) }
      selStart:  integer;      { début de sélection de texte }
      selEnd:    integer;      { fin }
      active:    integer;      { <>Ø si actif }
      wordBreak: ProcPtr;      { proc. de coupure de mot }
      clikLoop:  ProcPtr;      { Click loop rout. }
      { addresse d'une routine applée par TEClick }
      { pour du "scroll" de texte, par exemple }
      clickTime:longint;       { instant du premier clic }
      clickLoc: integer;       { Où le clic? }
      caretTime: integer;      { délai de scintillement }
      caretState: integer;
      just:      integer;      { mode }
      TELength: integer;       { nbre de caractères (long.) }
      hText: Handle;           { Handle sur le  texte }
```

```
    recalBack: integer;        { <>0 si rappel en background }
    recalLines: integer;       { Ligne rappelée }
    clikStuff: integer;        { click stuff (interne) }
    crOnly: integer;       { -1 si le retour chariot coupe un mot }
    txFont: integer;           { Font }
    txFace: Style;             { gras, standard,... }
    txMode: integer;           { Mode }
    txSize: integer;           { Taille }

    inPort: GrafPtr;           { le Grafport }
    highHook: ProcPtr;         { Highlighting hook }
    caretHook: ProcPtr;        { Highlighting hook }
    nLines: integer;           { Nombre de lignes }
    lineStarts: array [0..16000] of integer;
                               { débuts de lignes }
  end; { TERec }
```

ROUTINES DE TEXTEDIT

Utilisez **TEInit** pour initialiser TextEdit. Puis, créez une zone de travail:

TeNew(dest, view: Rect): TEHandle;

Cette fonction retourne un TextHandle sur le texte à saisir.

pour avoir un point d'insertion scintillant, placez:

TEIdle(h: TEHandle);

au début de la boucle d'événements. Si un mousedown arrive dans le viewrect (if ptinrect) d'un TERec (*inContent*), alors:

TEClick(pt: Point; extend: boolean; h: TEHandle);

modifiera éventuellement le texte et le sélectionnera. On aura d'abord fait passer les coordonnées du point de global à locales (GlobalToLocal(myEvent.where).

Lorsqu'un keyDown ou AutoKey se produit, on passe le code ASCII (dans le low-word de l'Event.message) à:

TEKey(key: char; h: TEHandle);

qui insère des caractères, et en efface par la touche BackSpace.

Entre BeginUpdate et EndUpdate, on place diverses routines, comme par exemple TEUpdate, TEActivate, TEDeactivate.

TEActivate(h: TEHandle);

active le scintillement du point d'insertion et "highlights" (inverse) les zones de texte précédemment sélectionnées.

TEDeactivate(h: TEHandle); fait exactement le contraire.

TEUpdate(rUpdate: rect; h: TEHandle); retracera le texte à votre place. Le texte sera ré-affiché depuis le point d'insertion dans le viewrect.

La procédure: **TECut**(h: TEHandle);

transfère la portion de texte sélectionnée vers le scrap de TextEdit scrap, et la retire du reste du texte.

TECopy(h: TEHandle);

transfère la portion de texte vers le scrap de TextEdit, sans la retirer du texte.

TEPaste(h: TEHandle);

insère le contenu du scrap de TextEdit à droite du point d'insertion.

TEDelete(h: TEHandle);

enlève du texte les caractères sélectionnés. Il ne les place pas dans le scrap de TE. On s'en sert dans un menu "Edition" où l'on placerait un item de menu "Effacer".

TEInsert insère le texte spécifié. Comme TEDelete, il ne modifie pas le scrap de TE. Tous deux peuvent être utilisé pour créer un item de menu "Annuler".

L'alignement du texte est effectué en se servant de:

TESetJust(just: integer; h: TEHandle);

Nous avons vu les constantes correspondant aux valeurs à utiliser (dans le champ *just*). Ensuite, vous appelerez TEUpdate afin de ré-afficher le texte avec le nouvel alignement.

Dans une boîte de dialogue, on peut vouloir sélectionner, du texte (comme un nom par défaut: "Sans titre"). Il faudra utiliser:

TESetSelect(selStart, selEnd: LongInt; h: TEHandle);

Lorsque selStart est égal à selEnd, seul s'affiche le point d'insertion. Ressemble au SeliText du Dialog Manager.

Si, ayant deux textes différents affichés dans deux fenêtres séparées, vous voulez garder ces textes dans la même zone, vous servant du même TEHandle, vous appelerez TESetText:

TESetText(text: Ptr; length: LongInt; hTE: TEHandle);

et agirez comme si vous aviez deux zones de travail distinctes. Ensuite, appelez TEUpdate.

TECalText vous aidera à recalculer les débuts de lignes dans le TERec, et mettra à jour les éléments du tableau lineStart.

TECalText(hte: TEHandle);

La fonction **TEGetText**(hte: TEHandle): CharsHandle;

retourne un handle vers le texte du TERec. Le champ teLength vous indiquera la longueur du texte en nombre de caractères.

Vous en avez terminé avec le texte?

TEDispose(h: TEHandle);

libèrera la place occupée par le handle (ou plutôt l'objet sur lequel il double pointe), et le texte s'en ira du heap à la prochaine compaction de la mémoire.

Il y a d'autres routines dans TextEdit, mais le peu que nous en avons vu nous suffit pour rentrer dans la programmation du Mac.

AINSI FONTS FONTS FONTS LES

POLICES DE CARACTERES...

Voici le Font Manager organisé, comme tous ses camarades Manager, autour de son record, appelé FontRec. Nous appelerons par la suite FM le Font Manager, et remplacerons les termes "police de caractères" par le mot "font", qui signifie la même chose en anglais, et en beaucoup plus court.

Une font est une ensemble de symboles graphiques - jusqu'à 255 par font. Ces symboles peuvent avoir n'importe quelle forme, et même celle... de caractères.

Une font a son numéro de font (**font number**), donc on peut aller la chercher dans un fichier des ressources (celui de l'application, du document ou du système). La font a aussi un nom et une taille. Celle-ci est exprimée en points. Un point, c'est 1/72 de pouce.

La taille correspond à la distance (en points) entre deux **ascent lines** (reportez-vous au chapitre IV où se trouve la Fig. explicative).

Le standard est de 80 points par pouce, approximativement la résolution du Macintosh™. QuickDraw transfère de l'information vers le Font Manager, et réciproquement. Le FM lit les fonts depuis le fichier des ressources, et effectue les changements d'échelle requis lorsque la font est disponible dans une taille différente de celle spécifiée.

Si vous vous servez de l'utilitaire Font DA Mover, et que vous repérez une font d'une certaine taille, dans le fichier System, passez de suite à MacWrite et utilisez cette font dans une taille qui n'existe pas dans le système. Et voyez les résultats que vous obtenez. Parfois, ce n'est pas très beau, mais c'est toujours le produit de l'action du FM.

La taille maximum admise est de 127 points, et pour être lisible, une font ne devrait pas avoir une taille inférieure à 6 points. L'allocation maximum de taille est de 32 Koctets pour une font.

En plus du numéro de font, et de la taille, le FM passe les informations suivantes à QuickDraw:

- style,

- facteurs d'échelle horizontal et vertical (numérateur et dénominateur: un numérateur de 1 et un dénominateur de 2 signifient que la font sera doublée),

- un booléen indiquant si oui ou non le caractère doit être tracé. Par exemple, si la fonction CharWidth est appelé, nul besoin de tracer, puisque cette fonction ne fait que mesurer les caractères,

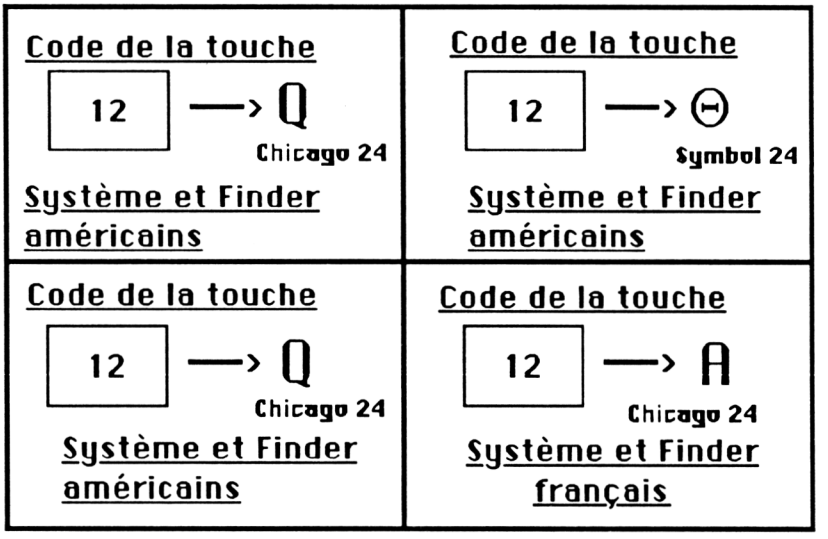
- un numéro spécifiant le périphérique de sortie. Ø, c'est l'écran du Mac.

Le FM a quelques numéro de fonts pré-définies. En voici la liste. Elle débute par Chicago (no de font:Ø). Une application se servira des fonts de l'application. Si rien n'est spécifié, c'est Genève qui est prise par défaut.

```
const
    systemFont  =  0;
    applFont    =  1; { font de l'application}
    newYork     =  2;
    geneva      =  3;
    monaco      =  4;
    venice      =  5;
    london      =  6;
    athens      =  7;
    sanFran     =  8;
    toronto     =  9;
```

Avant le Mac Plus, le clavier américain avait 58 touches, et le clavier européen en comportait 59. L'assignation des codes de touches est la même pour tous les claviers. C'est au moment de l'initialisation du système (au chargement) que chaque touche reçoit son code caractère. Et finalement, à un code d'une font est associé un symbole.

La figure qui suit apportera un peu de lumière sur ces assignations successives:



Le code de la touche, nous savons où le trouver si nous en avons besoin: c'est la partie de gauche du champ event.message (**div** 256. Voir le chapitre sur "l'Event Manager").

ECHELLES DE FONT

Le FM cherche la font de la taille demandée dans le fichier des ressources du document. S'il ne la trouve pas, il va dans les ressources de l'application, et enfin dans les ressources du système. S'il la trouve dans un de ces fichiers des ressources, rien ne se passe, sinon, le FM répète cette série de recherches pour la font de taille double, et il effectuera une mise à l'échelle avec un facteur de 1/2.

S'il ne trouve pas de font de taille double à celle requise, le FM cherchera une taille plus importante, et effectuera la mise à l'échelle - réduction. Sinon, il cherchera la font de taille immédiatement inférieure, et effectuera alors la mise à l'échelle supérieure.

Si, par contre, c'est le nom ou le numéro de la font qui ne sont pas trouvés, c'est la font de l'application (Genève par défaut), qui sera prise et mise à l'échelle.

Enfin, c'est la font système qui sera utilisée en dernier recours. Et c'est sur elle que sera effectuée la mise à l'échelle.

FONT RECORD

```
FontRec = record
      fontType:     integer; { type }
      firstChar:    integer; { code ASCII du 1er caractère }
      lastChar:     integer; { code ASCII du dern. caractère  }

      widMax:       integer; { largeur maximale de caractère }
      kernMax:      integer; { "kern" du caractère * -1 }
     { le "kern" du caractère, c'est la possibilité - en points - }
     { de recouvrit partiellement une autre lettre }
      nDescent:     integer; { descent * -1}
      fRectWidth:   integer; { largeur du font rectangle }
      fRectHeight:  integer; { hauteur du font rectangle }
      owTLoc:       integer; { déplacement depuis }
     { la table de déplacement/largeur }
      ascent:       integer;
      descent:      integer;
      leading:      integer;
      rowWords:     integer; { row width du bit image / 2 }
{     revoyez la section consacrée au GrafPort (chapitre IV)
pour plus de détails sur la largeur du caractère, les champs
ascent, descent, etc.  }
      { bitImage:array[1..rowWords,1..chHeight] of integer;
      locTable:     array [firstChar..lastChar+2] of integer;
      owTable:      array [firstChar..lastChar+2] of integer; }
    end;
```

Le type de font sera fixe ($BØØØ), ou proportionnel ($9ØØØ). C'est alors que sera ou non pris en considération l'espace supplémentaire (spExtra).

QUICKDRAW ET LE TEXTE...

Ce n'est pas une histoire qui ressemble à celle de Pierre et le loup que nous annonçons à présent. Seulement quelques mots sur ce qu'est le texte aux yeux de QuickDraw, et comment il le

manipule.

Un élément de font est en soi un bitimage. Un bitimage peut être copié sur un autre bitimage. Et il peut être placé à gauche ou à droite du crayon graphique, dont nous pouvons fixer la position. C'est presque tout. Ajoutons encore que lorsque le caractère frappé ne se trouve pas dans la font, c'est le carré qui s'affiche ("symbole manquant").

Après avoir tracé le caractère, QD déplace le crayon graphique du nombre de pixels correspondant à la largeur du caractère.

Le caractère espace est tracé un peu différemment, en fonction de la valeur du premier champ du fontrec, et d'autres facteurs qui dépendent de l'application et des instructions que QD doit exécuter (par exemple, en cas de justification à gauche et à droite, non supportée par TextEdit).

ECHANGE DE DONNEES AVEC QUICKDRAW

Avant de vous présenter les quelques routines du FM, voici deux records par lesquels transite l'information vers et depuis QD. FMInput, c'est l'information échangée depuis QD vers le FM. Devinez ce qu'est FMOutput.

```
type
 FMInput = packed record
        family:   integer;        { no de font }
        size:     integer;
        face:     Style;
        needBits:boolean;         { sera ou non dessiné }
        device:   integer;        { no réf. périph. dans l'octet }
                                  { le plus significatif }
        numer:    Point;          { numérateur }
        denom:    Point;          ( dénominateur; des points. }
       { le ratio numer.v div denom.v donne le facteur vertical  }
       { le ratio numer.h div denom.h donne le facteur horizont. }
      end;

 FMOutPtr = ^FMOutPut;

FMOutPut = packed record
        errNum:       integer;  { inutilisé }
```

```
  fontHandle:  Handle;       { sur le FontRec }
  bold:        Byte;         {fact. gras }
  italic:      Byte;
  ulOffset:    Byte;         { ul = underline }
  ulShadow:    Byte;
  ulThick:     Byte;
  shadow:      Byte;
  extra:       SignedByte;   { largeur du style }
  ascent:      Byte;         { asc. }
  descent:     Byte;
  widMax:      Byte;         { larg. max. du caract. }
  leading:     SignedByte;
  unused:      Byte;
  numer:       Point;
  denom:       Point;
end;
```

Les champs allant de "bold" à "shadow" indiquent les modifications de style. Il est important de savoir que tous les caractères/symboles d'une font n'ont pas **nécessairement** la même largeur.

ROUTINES DU FONT MANAGER

Immédiatement après InitGraf(@thePort), appelez **InitFonts**.

Vous savez qu'il vous est possible de placer dans vos menus les fonts dont vous disposez, tout comme les accessoires de bureau. Nous avons évoqué ce point lors de l'affichage des menus. AddResMenu fait ceci dans MacWrite avec les ressources de type 'FONT'. Puis, vous pourrez utiliser:

GetFontName(fontNum: integer; **var** theName: Str255);

Si la font est trouvée, thename reçoit le nom correspondant associé au no de font. Sinon, thename reste vide. Son symétrique:

GetFNum(fontName: Str255; **var** theNum: integer);

Si la font n'est pas trouvée, thenum est mis à Ø.

La fonction:

realFont(fontNum: integer; size: integer): boolean;

retourne TRUE si la font qui correspond au numéro de font fontNum existe dans la taille spécifiée, et FALSE si une mise à l'échelle doit être effectuée.

La procédure suivante vous permet de garder en mémoire la font la plus couramment utilisée dans une application. La font ne sera pas purgée du heap si lockFlag est mis à TRUE.

SetFontLock(lockFlag: boolean);

Nous terminons ici l'étude du Font Manager, et vous disons à bientôt pour programmer ensemble d'autres petites applications.

CHAPITRE XII

UN TRAITEMENT DE TEXTE, DEUX FENETRES

Viennent à présent les exemples de programmes. Ils comprennent des annotations. Le premier programme présenté ici est le noyau d'un traitement de texte. Il fait tout ce que fait une telle application, joue avec les accents, le backspace, les touches option et Commande, tout vous dis-je sauf l'usage des tabulations et la double justification.

De plus, il peut travailler sur **deux** fenêtres de texte, ce qui n'a été montré dans aucun des exemples fournis par Apple. Ces deux fenêtres peuvent aller jusqu'à une certaine limite, et on ne peut les "entrainer" trop loin en les transportant.

Dans cet exemple, la boucle d'événements est **complète**. Entrons dans ce programme sans plus tarder.

Commençons donc par voir les ressources de ce programme:

c'est un bon moyen de voir les types de ressources existant, et souligne un point important: votre programme doit être bien tracé dès que vous commencez le codage; en effet, celui-ci débute par l'écriture des ressources du programme.

Monprog.Rsrc

```
Type MENU
 ,128
* MENU # 128
 \14
*** the apple symbol ***
   A Propos de Monprog...^2<O
* A Propos..., une icone no 258, outline.

 ,129
 MyFile
* MENU # 129
   Quit^1<U/Q
*   "Quit" item de menu no 1
* Une icone, soulignée, un raccourci (Commande-Q)

 ,130
* MENU # 130
 YourEdit
   (Undo<B/Z
* Caractères gras. Command-Z déclenchera le Undo
   (-
* ligne de pointillés...
   Cut/X
   Copy/C
   Paste/V
   Clear

Type WIND
* première fenêtre, no 128
 ,128
 First Sample
* titre
 35 40 250 350
* coordonnées des positions des coins.
 Visible NoGoAway
 4
* type de fenêtre (pour renseigner QD)
 0
```

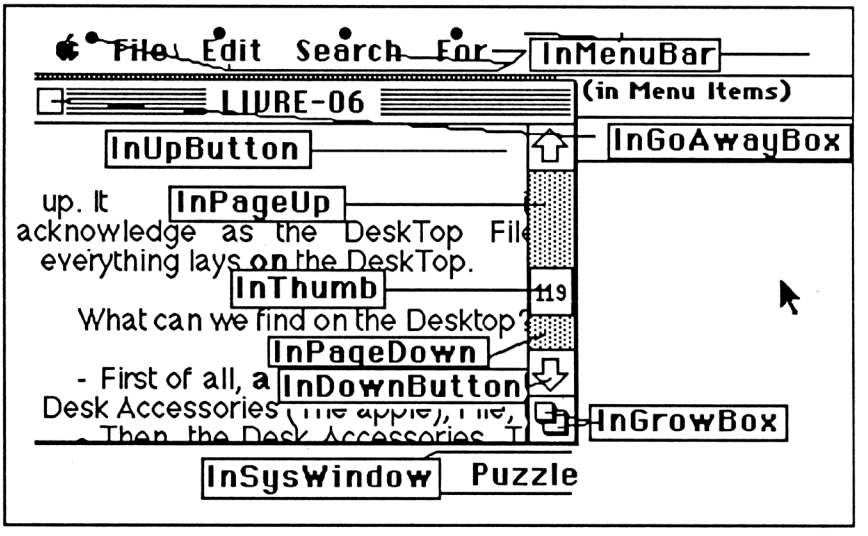
```
* proc. par défaut

   ,129
* 2è fenêtre, no ID 129.
  YANN Sample
  60 50 300 385
  Visible NoGoAway
  4
  0

  Type ICON
     ,257 (32)
* ICONE no 257, purgeable
     001F FE00
     0020 0000
     0020 F900
     0020 0500
     0020 0500
     0020 0500
     0020 0500
     0220 0500
     0420 0500
     1B20 0500
     24A1 F900
     2020 0100
     2020 0100
     1000 7900
     0E00 0100
     007C 0000
     0780 0000
     0807 FFC0
     07BF FFE0
     1FDF FDF8
     1FAA ABF8
     3FD5 55FC
     3FAA ABFC
     3FD5 55FC
     3FFA AFFC
     1FFF FFF8
     1FFF FFF8
     0FFF FFF0
     07FF FFE0
     01FF FF80
     003F FC00
     0000 0000
```

```
* définition de l'icone en hexadécimal

.    ,258 (32)
* autre icone, no ID 258
     0000 0000
     0000 0000
     000F 0000
     001B 0000
     0031 8000
     0060 C000
     0040 4000
     0FC0 67E0
     0880 3E30
     7F80 23BC
     C100 10E3
     8D00 1F19
     BD00 11CD
     E300 1867
     E200 0832
     D200 081E
     4A0C 3807
     47F2 6FFD
     4003 C0C1
     07BF FFE0
     1FDF FDF8
     1FAA ABF8
     3FD5 55FC
     3FAA ABFC
     3FD5 55FC
     3FFA AFFC
     1FFF FFF8
     1FFF FFF8
     0FFF FFF0
     07FF FFE0
     01FF FF80
     003F FC00
*    FFFF FFFF
*    FFFF FFFF ...
* (le masque n'est pas inclu)

*Curseur en X
Type CURS
     ,256
38383C7802800100010001000100010001000100010001000100010002803C
783838
```

* définition du curseur
38383C7802800100010001000100010001000100010001000100010002803C
783838
* Masque
0008 0008
* HotSpot

Type CODE
    Monprog,0

Nous avons vu les ressources. Passons au programme. Mais auparavant, il y a lieu de revoir les noms des endroits où se trouve la souris, après un FindWindow:



```
PROGRAM Monprog; { YANN FAIN, 15/5/86 }

 { Dessine deux fenêtres, et écritdu texte à l'intérieur }
{ vous pouvez passer du texte d'une fenêtre à l'autre, et
vice-versa, couper, copier, coller }

{-------------------------------------------------------------}
{$X-} { dé-active l'expansion du stack - LisaPascal }
{$U-}

USES
   {$U Obj/MemTypes } MemTypes,
   {$U Obj/QuickDraw} QuickDraw,  { (sous Workshop) }
   {$U Obj/OSIntf   } OSIntf,
   {$U Obj/ToolIntf } ToolIntf;
{-------------------------------------------------------------}
CONST

   appleID = 128;     { no ressource des menus }
   fileID     = 129;
   editID     = 130;

   appleM  = 1; { index pour myMenus, table de menuHandle }
   fileM      = 2;
   editM      = 3;
   LastMenu      = 3; { nombre de menus sur la barre }
   MenuBase      = 127; { voir l'explication dans }
                        { le premier exemple }
    windowID     = 128;    { n° des ressources des fenêtres }
    windowI1     = 129;
    undoCmd      = 1;       { n° d'item de menu }
    cutCmd       = 3;
    copyCmd      = 4;
    pasteCmd     = 5;
    clearCmd     = 6;
{-------------------------------------------------------------}
VAR
   myMenus: ARRAY[1..LastMenu] OF MenuHandle;
   { table de handles sur les menus. }
   { Nous avons ici trois MenuHandles. Les placer en table
permet un traitement plus général  (For i := 1 to LastMenu...) }

   dragRect, dra, drb: Rect;
   { rectangles servant de limites aux fenêtres }
   txRect,txa,txb: Rect;
```

```
    { rectangles de texte dans les fenêtres }

    textH,tHa,tHb:TEHandle; { handle sur le texte }
    theChar: CHAR;    { caractère frappé }
    extended: BOOLEAN;  { TRUE si l'utilisateur appuie sur }
                        { majuscule et clique en même temps }
    doneFlag: BOOLEAN;  { le programme lui donne la valeur }
                        { TRUE lorsqu'on a choisi "Quit" }
                        { il a reçu la valeur FALSE avant la
                        boucle d'événements }

    myEvent: EventRecord;

    wRecord,wRa,wRb: WindowRecord;
    { stocke l'information sur les fenêtres de cette application }

    myWindow,myWa,myWb, whichWindow: WindowPtr;
    { pointeurs sur des fenêtres }
    MyPeek, MyPeekA, myPeekB: windowPeek;
    { pointeurs sur des windowrecords }

    MyGa,MyGb, MyG1: GrafPtr; { Pointeurs sur des GrafPorts }
    { Notez bien que le premier champ d'un windowrecord }
    { et d'un dialogrecord est...un GrafPort; }
    { le port courant d'un windowrecord a la même adresse }
    { que le windowrecord en personne  }

    APtr: Ptr; { ayez toujours un Ptr a proximité }
    { (chez vous, au bureau, en voiture, etc.) }

    iBeam : CursHandle;
    { pour passer au curseur I-beam }

    { fin des définitions de variables }

{-------------------------------------------------------------}
PROCEDURE FixMenus;
{ Installe les menus et dessine la barre de menus }
{--------------------}

    { Les exemples fournis par Apple fournissent cette }
    { procédure écrite différemment }
    { Ils écrivent deux boucles: une qui va chercher les }
    { menus depuis le fichier des ressources }
    { la seconde pour insérer les menus dans la barre de menus }
```

```
VAR i: integer;

BEGIN
  { lire les menus depuis le fichier des ressources }
  { et stocker les menuHandles dans la table myMenus }

FOR i:=1 TO LastMenu DO
  BEGIN
    MyMenus[i]  :=GetMenu(i+MenuBase); { 127 + i }
    InsertMenu(myMenus[i],0);
  { installe chaque menu dans la barre de menus }
    IF i = 1 THEN
          AddResMenu(myMenus[1],'DRVR');
             { ajout des accessoires de bureau }
             { dans le menu pomme }
  END;    { FOR }
DrawMenuBar; { affichage de la barre de menus }
END;       { FixMenus }

{-------------------------------------------------------------}
PROCEDURE  ProperCursor;
{----------------------}
{ si on passe dans une fenêtre où du texte est écrit, }
{ changera le curseur flèche en curseur I-beam }
{ et vice-versa. }

VAR    mousePt : Point;
   { position de la souris }
BEGIN
  GetMouse (mousePt);
    { récupère les coordonnées de la souris }

    IF myWindow = FrontWindow THEN
    BEGIN
         IF PtInRect (mousePt,txRect) THEN
         { si on est dans la fenêtre de texte, alors... }
         SetCursor (iBeam^^)
    ELSE
         SetCursor (arrow);
    END; {* myWindow *}
END; {* CursorAdjust *}
{-------------------------------------------------------------}
  PROCEDURE SelWind;
{--------------------}
```

```
{ sélection de fenêtre }
BEGIN
   SelectWindow(whichWindow);
   { whichwindow est retourné par FrontWindow, }
   { et vous fait savoir où on a cliqué dans la fenêtre }
   { Ensuite, on le compare aux constantes fournies par }
   { le Toolbox Interface }

   { If window A est égale à whichwindow, on donne }
   { les zones spécifiques à la fenêtre A }
   { aux zones de transit }

   IF myWa = whichWindow THEN
   BEGIN
        myWindow:= myWa;
        wRecord := wRa;
        myPeek  := myPeekA;
        textH   := tHa;
        txRect  := txa;
        MyG1    := GrafPtr(myWa);
        END
        ELSE
        { Si ce n'est pas la fenêtre A, c'est donc la fenêtre B }
        { puisqu'on n'a que deux fenêtres d'application }
        BEGIN
        myWindow:= myWb;
        wRecord := wRb;
        myPeek  := myPeekB;
        textH   := tHb;
        txRect  := txb;
        MyG1    := GrafPtr(myWb);
        END;
    SetPort(whichWindow);
   { On fixe le port courant }
END;
```

```
{-------------------------------------------------------------}
PROCEDURE ExecCmd (theResult: LONGINT);
{-----------------------------------}
{ Exécute les commandes correspondant à l'item de menu }
{ retourné par menuselect (theResult). Sur des items en grisé }
{ ou en dehors du menu,  theResult est égal à zero. }

VAR
   theItem: INTEGER;
   { numéro d'item; contiendra le low word de theresult }

   theMenu: INTEGER;
   { numéro de menu }

   name: Str255;
   { nom de l'accessoire de bureau à passer au GetItem }
   temp: INTEGER;

   BEGIN
   theItem := LoWord(theResult);

   theMenu := HiWord(theResult);

   CASE theMenu OF  { dépend du numéro de menu }

{ ********* }
  appleID:
{ ********* }
   BEGIN
   GetItem(myMenus[appleM],theItem,name);
   { récupère l'accessoire de bureau }
   temp := OpenDeskAcc(name);
   { l'exécute }

   SetPort(myWindow);
   { restaure le port courant. }
   END;   { appleID }

{ ********* }
  fileID:
{ ********* }
   doneFlag := TRUE;
   { Quit. La boucle d'événements s'exécute jusqu'à ce que }
   { doneFlag soit FALSE }
{ ********* }
```

```
  editID:
{ ********* }
    BEGIN    { appel au Desk Manager pour gérer l'édition }
    IF NOT SystemEdit(theItem-1)
    { si l'accessoire de bureau est actif }
          THEN    { fenêtre d'application active }
          CASE theItem OF
          { numéro d'item }
          cutCmd: TECut(textH);
          { appel à TextEdit }
          copyCmd: TECopy(textH);
          pasteCmd: TEPaste(textH);
          clearCmd: TEDelete(textH);
       END;   { item case}
    END;      { editID}

  END; { menu case}

   HiliteMenu(0);        { Menu Manager pour restaurer }
    { l'inversion provoquée par MenuSelect)}
END; { ExecCmd}

{-------------------------------------------------------------}
BEGIN  {  phase principale d'appel }
{--------------------}

{ Initialisations }
Moremasters;
InitGraf(@thePort);    { init. QuickDraw}
InitFonts;             { init. Font Manager}
InitWindows;           { init. Window Manager}
InitMenus;             { init. Menu Manager}
TEInit;                { init. TextEdit}
InitDialogs(NIL);      { init. Dialog Manager}
InitCursor;            {  curseur flèche }
FixMenus;              { installation des menus }

WITH screenBits.bounds DO
{ fixe les limites de transport de fenêtres }
    SetRect(dragRect,5,25,right-5,bottom-5);
    { au moins 5 pixels de chaque côté de l'écran }
    { l'utilisateur ne peut ainsi perdre le contrôle }
    { de sa fenêtre. Un menu occupe 20 pixels en hauteur, }
    { donc 20 + 5 = 25 }
```

```
    doneFlag := FALSE; { sera à FALSE dès que l'on }
    { cliquera sur Quit }
{-------------------------------------------------------------}
myWindow := GetNewWindow(windowID,@wRa,POINTER(-1));
{ on lance la fenêtre }
SetPort(myWindow); { rend cette fenêtre le port courant }

{ initialisation des zones de cette fenêtre }
myWa  := myWindow;       { windowPtr }
myGa  := GrafPtr(myWa);  { GrafPort }
myPeekA:= POINTER(myWindow); { windowPeek }
myPeekA^.windowkind:= 8;
{ fenêtre d'application (non utilisé dans ce prog. sauf pour }
{ signaler qu'on peut s'en servir) }

txRect   := MyGa^.portRect;
{ rectangle pour le texte dans la fenêtre }
InsetRect(txRect,4,0);
{ fixe les limites à 4 pixels à l'intérieur }
txa:=txRect;
{ rend txa, spécifique à la fenêtre 128 (fen. A), égal à }
{ txRect, rectangle général (zone de transit) }

textH := TENew(txa,txa);
{ TextEdit est prêt à recevoir du texte }
tHa:=textH;
{ sauve le texthandle général dans tHa }

{-------------------------------------------------------------}
{ identique à A, mais pour la fenêtre B }
myWindow := GetNewWindow(windowI1,@wRb,POINTER(-1));
SetPort(myWindow);
myWb:= myWindow;
myGb:= GrafPtr(myWb);
myPeekB:= POINTER(myWindow);
myPeekB^.windowkind:= 8;
txRect := MyGb^.portRect;
InsetRect(txRect,4,0);
txb:=txRect;
textH := TENew(txb,txb);
tHb:=textH;
{-------------------------------------------------------------}
iBeam:= POINTER (GetCursor (256));
{ va chercher le curseur no 256 dans les ressources }
```

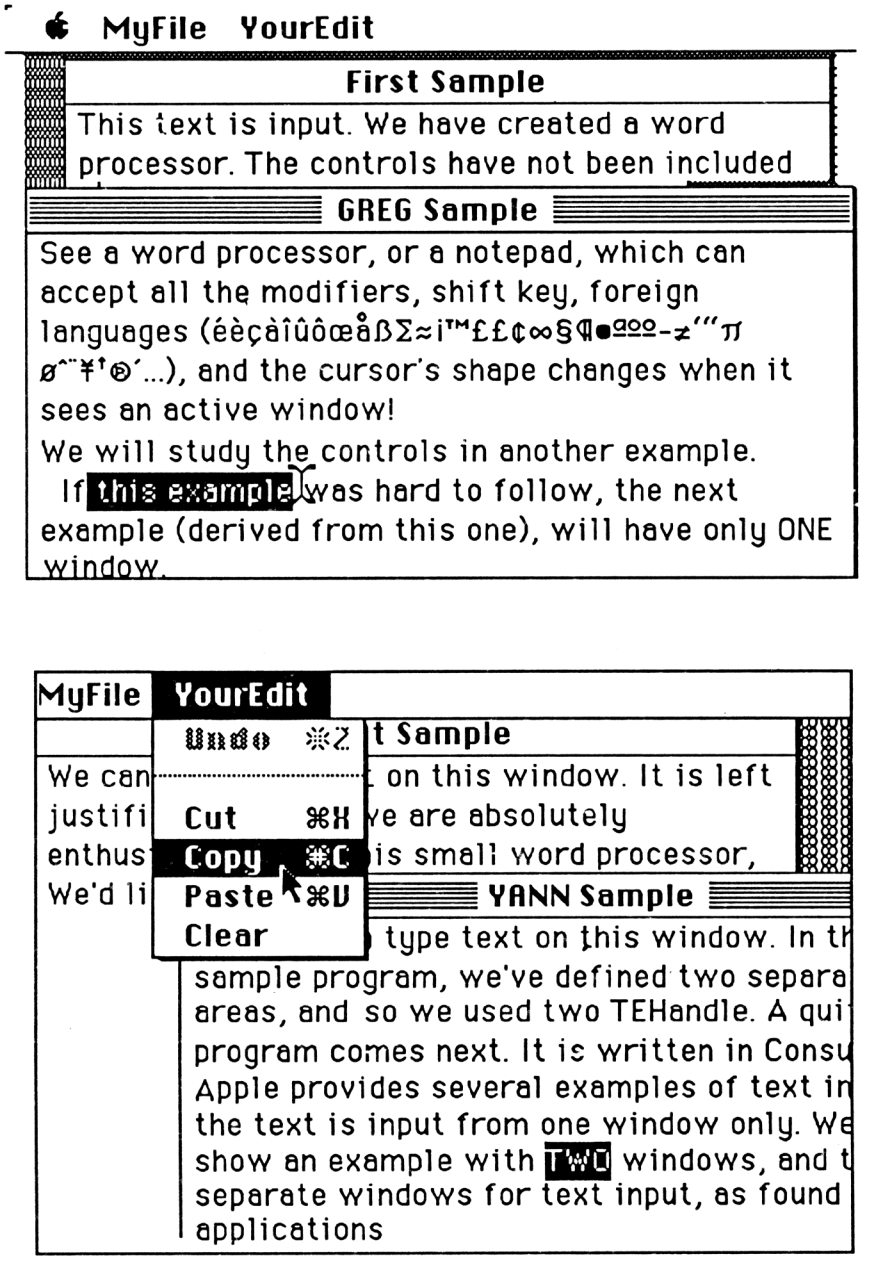
```
whichwindow := myWa;
FlushEvents(everyEvent,0);    { enlève tous événements }
                              { de la file d'attente des événements }
SelWind; { crée un activate qui amène la fenêtre A }
{ au premier plan }

{ boucle d'événements }

REPEAT
   ProperCursor;      { cette routine ajuste la forme du curseur }
{ selon l'endroit où se trouve la souris }

   SystemTask;
   { Appel au Desk Manager pour init. périodique }
   TEIdle(tHa);  { barre verticale de fen. A scintille }
   TEIdle(tHb);  { barre verticale de fen. B scintille }
{-------------------------------------------------------------}

   IF GetNextEvent(everyEvent,myEvent)
   { tous types d'événements sélectionnés }

   THEN

    CASE myEvent.what OF
   { dépendemment du type d'événement }

{ ********************* }
    keyDown, autoKey:          { touche appuyée ou }
{ ********************* }    { maintenue enfoncée (répét.) }
     BEGIN
   theChar := CHR(BitAnd(myEvent.message,charCodeMask));
                  { récupère le caractère }

        IF BitAnd(myEvent.modifiers,cmdKey) <> 0
        { si la touche Commande est aussi appuyée }
        THEN ExecCmd(MenuKey(theChar))
        { c'est un raccourci. On va dans les menus  }
        ELSE TEKey(theChar,textH);
        { Autrement, on livre le caractère à TextEdit }
   END;

{ ********************* }
    mouseDown:
   { clic. Appel au WM pour savoir où on a cliqué... }
{ ********************* }
```

```
    CASE FindWindow(myEvent.where,whichWindow) OF

{ ********************* }
    inSysWindow:
{ ********************* }
      { dans les accessoires de bureau. Le Desk Manager }
       { s'en occupe }
      SystemClick(myEvent,whichWindow);

{ ********************* }
    inMenuBar:
{ ********************* }
      ExecCmd(MenuSelect(myEvent.where));
       { dans la barre de menus. Le paramètre myEvent.where }
       { sera "theResult" }

{ ********************* }
    inDrag:
{ ********************* }
       { dans la barre de titre }
       DragWindow(whichWindow,myEvent.where,dragRect);

{ ********************* }
    inContent:
{ ********************* }
  { Ici se trouve le corps de votre application }
  { les lignes de code que l'on écrit dépendent de ce que fait }
  { l'application. Il existe aussi des applications sans fenêtre }
  { mais qui comportent des dialogues }
  BEGIN  { inContent }
      IF whichWindow <> FrontWindow
      { si la fenêtre n'est pas active (FrontWindow) }
      THEN SelWind
           { on la rend active, }
      ELSE
      { sinon }
      BEGIN
        GlobalToLocal(myEvent.where);
        { les coordonnées de la souris deviennent locales }

        extended := BitAnd(myEvent.modifiers,shiftKey) <> 0;
        { test pour majuscules; extended est un booléen }

        TEClick(myEvent.where,extended,textH);
       { TextEdit le traite }
```

```
        END;
      END;   { inContent }
    END;   { mouseDown}

{ ********************* }
   activateEvt:
{ ********************* }
    BEGIN
    IF BitAnd(myEvent.modifiers,activeFlag) <> 0
  { c'est un activate }
      THEN
       { la fenêtre d'application devient active }

       BEGIN
        TEActivate(textH);
        { Highlight du texte sélectionné - s'il y en a }
       { et restaure la barre verticale scintillante au point }
       { d'insertion }
        DisableItem(myMenus[editM],undoCmd);
        { Dé-active Undo: cette application ne le gère pas }
       END
      ELSE
       { c'est un dé-activate }
       BEGIN
       { la fenêtre d'application devient inactive }
       TEDeactivate(textH);
       { désélectionne le texte sélectionné s'il y en avait }
       { et retire la barre verticale scintillante }
       EnableItem(myMenus[editM],undoCmd);
       { restaure Undo: }
       { les accessoires de bureau la supportent }
       END;
    END;  { activateEvt}
```

```
{ ********************* }
    updateEvt:
{ mise à jour de fenêtre }
{ ********************* }
      BEGIN
       BeginUpdate(WindowPtr(myEvent.message));
          { déjà vu }
       SelWind;    { cette procédure }
          { rend la fenêtre active, sans générer d'événement }
       EraseRect(MyG1^.portRect);
          { appel à QuickDraw pour effacer ce qu'il y avait dans }
          { le portrect général }
       TEUpdate(MyG1^.portRect,textH);
       { Mise à jour du texte }
       EndUpdate(WindowPtr(myEvent.message));
       { le Window Mgr termine la mise à jour, et }
       { permet de nouveau la génération d'événements }
       END;   { updateEvt}

     END;   { event case}

UNTIL doneFlag; { on sort lorsque doneflag = TRUE }

{ c.a.d; lorsque l'utilisateur a relâché la souris sur  QUIT }
{ Aucune action n'est prise ici. Normalement, il faudrait }
{ afficher un dialogue demandant si l'on veut }
{ sauvegarder le texte }

END. { programme }
```

Examinons à présent les sorties des fenêtres, obtenues une fois encore en appuyant sur les touches "Verrouillage Maj.", la touche Command et le chiffre 3, qui produisent des dessins au format MacPaint, Screen Ø à Screen 9, pendant l'exécution de ce programme. On aurait pu procéder autrement pour saisir le texte: en n'utilisant qu'une zone de texte pour les deux textes, et un déplacement depuis le début du texte pour localiser celui de la deuxième fenêtre.


MyFile YourEdit
First Sample
This text is input. We have created a word processor. The controls have not been included
GREG Sample
See a word processor, or a notepad, which can accept all the modifiers, shift key, foreign languages (éèçàîûôœåßΣ≈i™££¢∞§¶•ºº-≠'"π ø^¨¥†®´...), and the cursor's shape changes when it sees an active window!
We will study the controls in another example.
If this example was hard to follow, the next example (derived from this one), will have only ONE window.
MyFile
YourEdit
Undo
Cut
Copy
Paste
Clear
t Sample
on this window. It is left
ve are absolutely
is small word processor,
YANN Sample
type text on this window. In th
sample program, we've defined two separa
areas, and so we used two TEHandle. A qui
program comes next. It is written in Consu
Apple provides several examples of text in
the text is input from one window only. We
show an example with TWO windows, and t
separate windows for text input, as found
applications

A présent, nous fournissons un deuxième exemple de programme, en "C" cette fois. Comme les instructions spécifiques au Mac sont similaires, sauf en ce qui concerne la syntaxe, les explications seront plus rares.

```
#define UseWithStdLib

/* Déclarations. Si vous utilisez la librairie standard, vous
devez inclure Stdio.h. Cet exemple est adapté d'un de ceux du
"C" de Consulair, avec la permission de Bill Duval */

#include "MacCDefs.h"

#ifdef UseWithStdLib
  #include "Stdio.h"
#else
  extern struct P_Str *CtoPstr();
  int strlen(str) char *str;
    {int i=0; while (str[i++]); return i-1;}
#endif

#include "Window.h"
#include "Events.h"
#include "TextEdit.h"
#include "Menu.h"

 /* Déclaré ici. Nous avons déjà fourni les explications dans
l'exemple de "C" précédent. */

  MenuHandle DeskMenu;
  MenuHandle EditMenu;
  MenuHandle Menu;

  #define Desk_ID 200
  #define Edit_ID 201
  #define Menu_ID 202
  Rect screenRect    = {0, 0, 384, 512};
  Rect windowRectA       = {50, 50, 200, 400};
```

```
    Rect windowRectB        = {60, 60, 210, 410};

    WindowPtr openWindow();

    #define False 0
    #define True 0xFF

   /* Déclarés ailleurs */

    #ifdef UseWithStdLib
      extern WindowPtr console; /* Std Lib TTY Window */
    #endif

/* Code */

   Init()
    {
    InitDialogs(0);
    TEInit();
    InitMenus();

    /* Desk Accessory menu */
      DeskMenu = NewMenu(Desk_ID,CtoPstr("\024"));
      AddResMenu(DeskMenu, 'DRVR');
      InsertMenu(DeskMenu, 0);

    /* Edit menu */
      EditMenu = NewMenu(Edit_ID, CtoPstr("Edit"));
      AppendMenu(EditMenu,
     CtoPstr("Undo;(-;Cut/X;Copy/C;Paste/V;Clear"));
      InsertMenu(EditMenu, 0);
      DisableItem(EditMenu, 0);

    /* "Menu" menu */
      Menu = NewMenu(Menu_ID, CtoPstr("Menu"));
      AppendMenu(Menu,
     CtoPstr("Item 1;(Dimmed Item 2;Item 3;(-;Item 5/5;Quit/."));
      InsertMenu(Menu, 0);

    DrawMenuBar();
    }

/* Tout ceci a déjà été commenté. Nous arrivons maintenant à la
partie principale de programme */
   main()
```

```
{
char c, *str;

short windowcode;
long menuResult;

EventRecord event; /* sans commentaire */

TEHandle hTE;
/* handle sur notre texte */
 WindowPtr mouseWindow, window, windowA, windowB;
/* nos fenêtres et leurs windowPtr respectifs */

if (CatchSignal()) ExitToShell();
/* retour au Finder si Signal détecté */
/* Equivalent en Pascal: doneflag:= TRUE */

Init();

#ifdef UseWithStdLib

 printf
("\r\r          Mac C \rCopyright Consulair Corporation \r     All
Rights Reserved\r\r");
 getchar();
/* cette application attend la frappe de texte */
 DisposeWindow(console);
/* adieu à la fenêtre TTY */

#endif

windowB =
 openWindow
(&windowRectB, " Fen. B","\r        Mac C Demo\r\251 Consulair Corporation
1984\r    All Rights Reserved\r\rThis is Window B");
/* ouverture de la fenêtre B sans ressources */
windowA =
 openWindow
(&windowRectA, "Fen.  A","\r        Mac C Demo\r\251 Consulair Corporation
1984\r    All Rights Reserved\r\rThis is Window A");
/* ouverture de la fenêtre A sans ressources */
hTE = 0;
/* null TEHandle;
InitCursor();
FlushEvents(0xFFFF);
```

```
   SelectWindow(windowA);
/* Génére un activate pour la fenêtre A */
/* notez que la plupart des initialisations sont effectuées
par le package */

/* boucle d'événements */

   while (True)
    {
    SystemTask();
/* déjà vu en Pascal */

    if (hTE)  TEIdle(hTE);
/* si texte, scintillement du point d'insertion */

    if (GetNextEvent(everyEvent, &event))
   {
   switch ( event.what )

    {
    case autoKey:
    case keyDown:
/* autokey et keydown - clavier */


      {
      c = event.message;
      if ((event.modifiers & cmdKey))
        DoMenu(MenuKey(c));
/* si la touche modifiers a été appuyée en même temps */
       else TEKey(c, hTE);
/* passe le caractère à textEdit */
      break;
      }

    case mouseDown:
/*------------------------------*/
      {
      windowcode = FindWindow(&event.where, &mouseWindow);
/* où a-t-on cliqué? */
      if (FrontWindow() != mouseWindow)
/* si la fenêtre cliquée n'est pas active, et si */
/* mouse window non nul, alors */
        if (mouseWindow != 0)
         {
```

```
        SelectWindow(mouseWindow);
/* la rendre active */
        break;
        }

      if ((window != 0) && (window == mouseWindow))
/* si le window pointer général est non nul, et si */
/* mouseWindow est non nul, alors */
        {
        if (mouseWindow != 0)
          {
/* en faire le port courant */
/* détail: on teste non nul car mouseWindow est à nul si on a
fermé cette fenêtre */
          SetPort(mouseWindow);
          switch ( windowcode )
           {
           case inContent:
            {
             GlobalToLocal(&event.where);
             TEClick(&event.where,
              (event.modifiers & shiftKey)? True:False, hTE);
/* cette construction de "C" évite un test if else */
             break;
            }
           case inDrag:
            {
/* transporte la fenêtre */
        DragWindow(mouseWindow, &event.where, &screenRect);
             break;
            }
           case inGrow:
            {
            long growResult;
            short vert, horiz;
            growResult =
             GrowWindow(mouseWindow, &event.where, &screenRect);
            horiz = growResult;
            vert = HiWord(growResult);
/* recalcule nouvelles coordonnées */
             SizeWindow(mouseWindow, horiz, vert, True);
             EraseRect(&mouseWindow->portRect);
/* efface le rectangle */
             InvalRect(&mouseWindow->portRect);
/* ajuste la taille du rectangle */
```

```
                SizeTE(mouseWindow);
                DrawGrowIcon(mouseWindow);
/* dessine l'icone de dimensionnement */
                break;
                }
              case inGoAway:
                {
                if (TrackGoAway(window, &event.where))
/* l'utilisateur a cliqué dans la boîte de fermeture */
                 {
                 TEDispose(hTE);
                 hTE = 0;
/* libère texte et handle */
                 DisposeWindow(window);
/* libère le handle */
                 window = 0;
/* pointeur mis à Ø */
                 }
                break;
               }
             }
           }
         }
        else
         {
         switch ( windowcode )
          {
          case inMenuBar:
           {
/* dans la barre de menu */
            DoMenu(MenuSelect(&event.where));
/* selon l'item de menu sur lequel */
/* le bouton de la souris a été relâché */
            break;
           }
          case inSysWindow:
           {
            SystemClick(&event, mouseWindow);
/* dans une fenêtre d'accessoire de bureau */
            break;
           }
          case inDrag:
           {
        DragWindow(mouseWindow, &event.where, &screenRect);
/* on transporte encore la fenêtre */
```

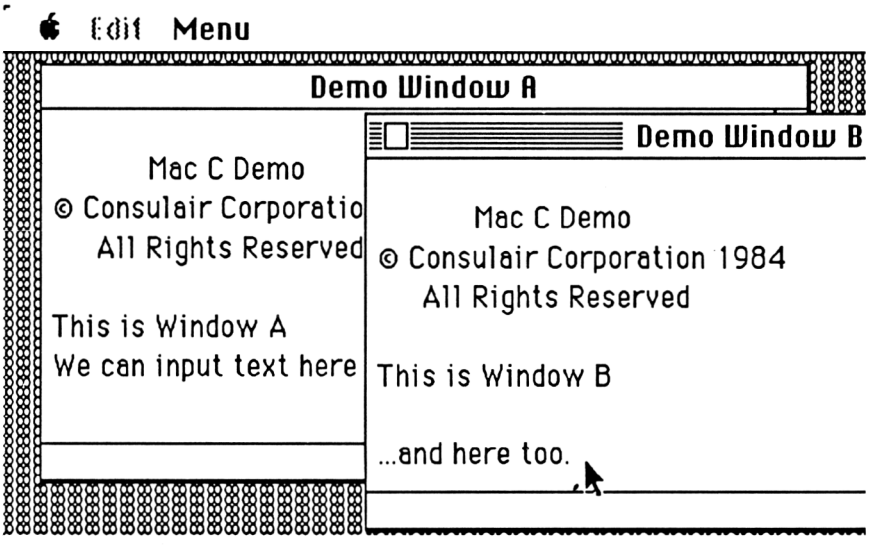
```
            break;
            }
           case inGoAway:
            {
            break;
            }
          }
        }
       break;
       }
      case updateEvt:
       {
       TEHandle temp_hTE;
       WindowPtr tempWindow;

       SetPort(tempWindow = (WindowPtr)event.message);
/* détypage. D'un côté, le champ event.message est converti en
windowptr et placé dans tempWindow */
/* ensuite, le SetPort assigne le port courant (actif) */
       BeginUpdate(tempWindow);
/* stoppe les événements */
       temp_hTE = (TEHandle)GetWRefCon(tempWindow);
/* fixe un TEHandle temporaire */
       TEUpdate(&tempWindow->portRect, temp_hTE);
/* restaure le texte dans la fenêtre appropriée */
       DrawGrowIcon(tempWindow);
/* dessine la boîte de dimensionnement */
       EndUpdate(tempWindow);
       break;
       }
/* cet exemple utilise deux boucles d'événements en une! */

      case activateEvt:
       {
       TEHandle temp_hTE;
       WindowPtr tempWindow;

       SetPort(tempWindow = (WindowPtr)event.message);
       temp_hTE = (TEHandle)GetWRefCon(tempWindow);
       if ((event.modifiers & activeFlag))
          {
          window = tempWindow;
          TEActivate(hTE = temp_hTE)
          }
         else TEDeactivate(temp_hTE);
```

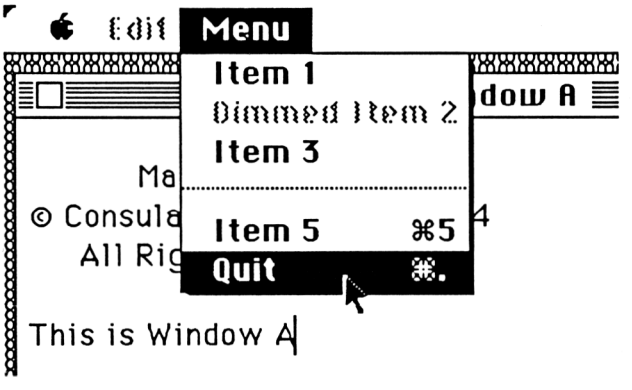
```
        DrawGrowIcon(tempWindow);
/* comme en Pascal */
        break;
        }
      }
    }
   }
  }

  DoMenu(menuresult)
/* exécute la commande sur laquelle on a relâché la souris */
    long menuresult;
    {
    short menuID, itemNumber;

    menuID = HiWord(menuresult);
    itemNumber = menuresult;
    switch ( menuID )
     {
     case Menu_ID:
    {
    switch ( itemNumber )
     {
     case 1: break;          /* item 1 */
     case 2: break;          /* item 2 */
     case 3: break;          /* item 3 */
     case 5: break;          /* item 5 */
     case 6:             /* item 6 = quit */
       Signal("All Done");
/* positionnement du Signal de fin */
     }
    break;
    }
     case Desk_ID:
    {
    struct P_Str AccessoryName;
    GetItem(DeskMenu, itemNumber, &AccessoryName);
    OpenDeskAcc(&AccessoryName);
    EnableItem(EditMenu, 0);
/* dé-active Undo */
    DrawMenuBar();
/* redessine la barre de Menu */
    break;
    }
```

```
  case Edit_ID:
  {
  SystemEdit(itemNumber-1);
  break;
  }
  }
  HiliteMenu(0);
 }

Rect *TERect(window, rect)
  WindowPtr window;
  Rect *rect;
  {
  BlockMove(&window->portRect, rect, sizeof(Rect));
  rect->right -= 16;
  /* on fait de la place pour la barre de défilement */
  rect->bottom -= 16;
  }
 TEHandle openTE(window)
  WindowPtr window;
  {
  Rect destRect, viewRect;
  TEHandle hTE;
  TERect(window, &viewRect);
  TERect(window, &destRect);
  destRect.left += 4;        /* on décale un peu */
  return TENew(&destRect, &viewRect);
  }

 WindowPtr openWindow(rect, title, str)
  Rect *rect;
  char *title, *str;
  {
  WindowPtr window;
  TEHandle hTE;
  window = NewWindow(0, rect, CtoPstr(title), True, 0, -1, True, 0);
  SetPort(window);
  hTE = openTE(window);
  SetWRefCon(window, hTE);
  TESetText(str, strlen(str), hTE);
  TEUpdate(&(*hTE)->viewRect, hTE);
  return window;
  }

  sizeTE(window)
```

```
WindowPtr window;
{
Rect rect;
TEHandle hTE;
hTE = (TEHandle)GetWRefCon(window);
TERect(window, &rect);
BlockMove(&rect, &(*hTE)->viewRect, sizeof(Rect));
rect.left += 4;          /* petit décalage */
BlockMove(&rect, &(*hTE)->destRect, sizeof(Rect));
TECalText(hTE);
}
```

A présent, après ce second exemple en "C", voici les résultats:





Si, après tout cela, vous n'avez pas de mal de tête, c'est que ce n'était pas si compliqué, après tout!

Nous avons renoncé à vous fournir un exemple de fichier EXEC sous Workshop. Au moment où vous lirez ces lignes, le système de développement Pascal d'Apple sera enfin sorti. Et il existe une profusion de bons "C" (celui de Think Technologies arrive, le Lightspeed "C"), et le Pascal TML. Donc, le Workshop (Mac XL) n'est plus aussi important qu'il le fut.

A présent, entrons dans les régions et les dessins animés.

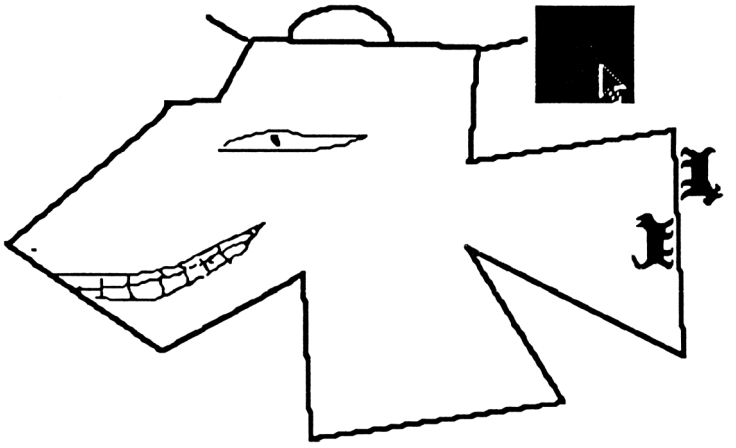
TRANSPORT DE REGIONS...

Le dernier des exemples va nous emmener au royaume des régions. Il montre comment des icones peuvent être amenées d'un endroit de l'écran à un autre. Transporter une icone, ou une region, c'est tracer du dessin et effacer la partie précédemment dessinée.

Donc, c'est aussi la base de tout **dessin animé**.

L'exemple qui suit fait tout cela, mais auparavant, il charge un dessin issu de MacPaint. On aurait très bien pu placer une PICT dans le fichier des ressources, ou la "Paster" au moyen de ResEdit. Ou encore, définir le dessin comme un PicItem placé dans un dialogue. En général, on peut toujours programmer la même application de diverses manières.

File Hello YANN



Le carré noir de cette figure est **transporté** au moyen de la souris.

Nous commencerons par lister les ressources du programme:

```
* demo Ressource file

demo.rsrc

Type MENU
  ,256
  File Hello YANN
    Quit

Type CODE
    demo,0
```

En fin de ce programme, une procédure dessine au hasard sur l'écran, et le programme ne retourne pas immédiatement au Finder. Il faut vivre dangereusement...

```
PROGRAM Demo;
{$U-}
{$X-}
{$D-}
{$R-}
{$M+}

{ "DEMO" de YANN Fain   06/1986       }
{ - Place un dessin MacPaint sur l'écran }
{ - Définit une Region }
{ - Transporte celle-ci au moyen de la souris }
{ - Définit une picture QUICKDRAW }
{   et la dessine au hasard sur l'écran  }
{   Ce programme peut être amélioré }
{   Considérez ceci comme un exercice }
{   La solution est d'utiliser la dernière sauvegarde de l'écran }
{   pour calculer la nouvelle, combinée avec }
{   celle de la nouvelle partie recouverte. }


USES
        {$U Obj/MemTypes  } MemTypes,
        {$U Obj/QuickDraw } QuickDraw,
        {$U Obj/OSIntf    } OSIntf,
        {$U Obj/ToolIntf  } ToolIntf;

const
```

```
  lastMenu  = 1;        (* nombre de menus *)
  FileMenu  = 256;      (* Menu File *)

var
  otherBits:packed array[1..512] of byte;
   { pour sauvegarde de l'écran }
  savBits:Bitmap; { de référence }
  error:Oserr;
  aPort: GrafPort;
  myMenus: array [1..lastMenu] OF MenuHandle;
  doneflag:boolean;

  PROCEDURE SetUpMenus;  { Initialise les Menus }

    VAR
      i: INTEGER;

    BEGIN
      InitMenus;
      myMenus[1] := GetMenu(fileMenu);
      FOR i := 1 TO lastMenu DO InsertMenu(myMenus[i],0);
      DrawMenuBar;

    END; { SETUPMENUS }

  PROCEDURE DoCommand(mResult: LongInt);
   (* nous avons déjà étudié ces procédures *)

    var
      theMenu,theItem:integer;

    BEGIN
    (* Which menu, Which command ? *)
    theMenu := HiWord(mResult);
    theItem := LoWord(mResult);

    CASE theMenu OF

        fileMenu:doneflag:=true; { une seule action: Quit }

        END; { menu case }

    HiliteMenu(0);

    END; { DOCOMMAND }
```

```
  procedure initall;
{ inits }

   begin
   FlushEvents(everyEvent,0);
   InitCursor;
   InitFonts;
   InitWindows;
   Setupmenus;
   Unloadseg(@SetupMenus);
   Openport(@aPort);
   end; (* initall *)

{ ********************************************************************** }

  procedure showpict;

   const
     rB=72;                (* rowBytes pour le Bitmap *)
     srcblocks=2;          (* taille du buffer E/Sr *)
     headersize=512;       (* taille du header pour fichier *)

   type
     diskBlock=packed array[1..512] of QDbyte;
     (* description du buffer E/S *)

   var
     srcbuf:array[1..srcblocks] of diskBlock;     (* buffer E/S. *)
     thebits:packed array[1..72] of byte;         (* Données *)
     srcPtr,dstPtr:Ptr;
     dstBits:Bitmap;
     refnum,scanline,srcSize:integer;
     count:longint;
     showrect,linerect:Rect;

   begin
   srcsize:=srcBlocks*512;
   srcPtr:=@srcBuf;
   (* Définition des paramètres du Bitmap *)

   dstBits.rowBytes:=rB;
   dstBits.baseaddr:=@Thebits;
   SetRect(dstBits.bounds,0,0,576,1);
```

```
  dstPtr:=POINTER(dstBits.baseAddr);

  (* Ouverture du fichier MacPaint *)

  error:=FSOpen('macPic',0,refnum);

  (* On saute le header, on moud l'café... *)

  count:=headersize;
  error:=FSRead(refnum,count,@srcBuf);

  (* Premier  read *)

  count:=srcSize;
  error:=FSRead(refnum,count,@srcBuf);

  (* on lit srcBlocks - 1 chaque fois *)
  count:=count-512;

{ Initialisation du rectangle associé au bitmap où est située }
{ la partie à afficher à l'écran }
  SetRect(linerect,0,0,512,1);

  (* Initialisation du rectangle de dessin à l'écran *)
  setRect(showrect,0,0,512,1);


  for scanline:= 1 to 342 do
   begin
    (* place les données du dessin MacPaint sur l'écran *)
    UnpackBits(srcPtr,dstPtr,rB);
    CopyBits(dstBits,screenBits,linerect,showrect,srcCopy,nil);
    (* préparation pour la partie suivante du dessin *)
    dstPtr:=pointer(dstBits.baseaddr);
    offsetrect(showrect,0,1);

    if (ord(srcPtr) > (ord(@srcBuf) + srcSize - 512))
     then
      begin
       srcBuf[1]:=srcBuf[srcBlocks];
       error:=FSRead(refnum,count,@srcBuf[2]);
       srcPtr:=POINTER(ord(srcPtr)+512-srcSize);
      end;
   end;
```

```
   (* fermeture du fichier MacPaint... *)
   error:=FSClose(refnum);
   end; (* SHOWPIC *)

{ ********************************************************************** }
  procedure mainloop;
   const
     maxp=40;
   var
     myEvent: EventRecord;
     whichWindow: WindowPtr;
     code:integer;

     Rgn,RgnDup,RgnUtil:rgnHandle;
     therect:rect;
     newloc,oldloc:point;
     go:boolean;

     rectutil,rcible:rect;

     temp: boolean;

     nbp:integer;
     ptab:array[1..maxp] of point;


{ ********************************************************************** }
   procedure saveB(freeRgn:Rgnhandle);

     (* Sauvegarde de ce qui est sur l'écran à l'endroit *)
     (* où se trouve le rectangle qui entoure la region Rgn. *)
     (* Nous l'utilisons avant de dessiner la region à *)
   { son nouvel emplacement }
     var
       srcRect,dstrect:rect;
       rsource:rect;

     begin
     srcRect:=Rgn^^.rgnBBox;
     dstrect.top:=0;
     dstrect.left:=0;
     dstrect.bottom:=srcRect.bottom-srcRect.top;
     dstrect.right:=srcRect.right-srcRect.left;

CopyBits(Theport^.portBits,savBits,srcRect,dstRect,srccopy,nil);
```

```
    rectutil:=dstrect;
    end;

{ ********************************************************************** }
    procedure restB(maskRgn:RgnHandle);

    { Redessine à l'endroit du rectangle entourant la  region Rgn }
     (* ce qui a auparavant été sauvegardé. *)
     (* Nous nous en servons avant de déplacer la région }
    { à un autre endroit }

    begin
    CopyBits(SavBits,Theport^.portBits,rectutil,rgn^^.rgnBBox,srccopy,nil);
    end;

{ ********************************************************************** }
    procedure preprgn;

     (* Définit la région: un petit carré. Après la sauvegarde *)
    { d'une partie de l'écran, on dessine cette région }

    begin
    Rgn:=NewRgn;
    OpenRgn;
    SetRect(therect,160,160,200,200);
    Framerect(therect);
    closergn(Rgn);

    (* 8 octets sur 64 lines, 512 bytes, ou 256 entiers *)
    savBits.BaseAddr:=@otherBits;
    savBits.rowBytes:=8;
    SetRect(savBits.bounds,0,0,63,63);
    SaveB(nil);
    fillRgn(Rgn,black);
    end; (* preprgn *)

{ ********************************************************************** }
    procedure endrgn;

     (* libère l'espace occupé par la région *)

    begin
    DisposeRgn(Rgn);
    end; (* endrgn *)
```

```
{ ********************************************************************** }
  procedure useQDpic;

  (* Génére une Picture qui est juste une partie du *)
  (*  trajet précédent de la souris, et la met au hasard *)
  { sur l'écran, en tailles diverses }
  { L'emploi du mode mode PatXor est un moyen simple de }
  { placer quelque chose sur l'écran et l'enlever avec la même }
  { séquence de dessin }

var
    pichndl:Pichandle;
    rec:rect;
    ti:longint;
    savepen:penstate;

  begin
  GetPenstate(savepen);

  (* crée la picture *)
  pichndl:=Openpicture(Theport^.portrect);
 { directement }
  Penmode(PatXor);
  PenPat(black);
  moveto(ptab[1].h,ptab[1].v);
  for i:=2 to nbp do lineto(ptab[i].h,ptab[i].v);
  ClosePicture;

  (* dessine la picture ici et là... *)
  for i:=1 to 35 do
    begin
    rec.top:=random div 100;
    rec.left:=random div 60;
    tai:=random div 200;
    rec.bottom:=rec.top+tai;
    rec.right:=rec.left+tai;
    (* l'affiche *)
    DrawPicture(pichndl,rec);
    (* temporisation *)
    ti:=tickcount;
    repeat until tickcount > ti+30;
    (* erase it *)
    DrawPicture(pichndl,rec);
  end;
```

```
  (* Laisse la mémoire en paix *)
  KillPicture(pichndl);

  SetPenstate(savepen);
 end; (* useQDPic *)

begin

InitGraf(@thePort);
preprgn;
doneflag:=false;
go:=false;
nbp:=0;

repeat

    temp := GetNextEvent(everyEvent,myEvent);

    CASE myEvent.what OF

      mouseDown:

        BEGIN
         code := FindWindow(myEvent.where,whichWindow);
         CASE code OF


            inMenuBar:
                 DoCommand(MenuSelect(myEvent.where));

             inSysWindow:
                 SystemClick(myEvent,whichWindow);

          inDrag:(**);       { vide }
          inGoAway:(**);
          inGrow:(**);
          inContent:(**);
          END;
         END; { mouseDown }

     keyDown,autoKey:(**);
     activateEvt:(**);
     updateEvt:(**);

    END; { case event of }
```

```
{ ********************************************************************** }
  { L'utilisateur appuie-t-il sur la souris avec le curseur sur }
  { la region? Si oui, transport de la souris et de la région
  if Button
   then
    begin
     GetMouse(oldloc);
     go:=PtinRgn(oldloc,Rgn);
    end;

  if go
   then
    begin
    repeat
       GetMouse(newloc);

  { mouvement suffisemment important  + - 5 pixels }
  { pour redessiner? }

  if ((abs(newloc.v-oldloc.v) >5) or (abs(newloc.h-oldloc.h) >5))
       then
        begin
         (* restaure l'écran *)
        RestB(nil);
         (* calcule le nouveau rect de la  region *)
         OffsetRgn(Rgn,newloc.h-oldloc.h,newloc.v-oldloc.v);

      (* sauvegarde de la partie de l'écran incluse dans *)
      { ce rectangle }
         SaveB(nil);

         (* dessine la region en noir *)
        fillRgn(Rgn,black);

        (* *)
        if (nbp<maxp)
          and
           ((abs(newloc.v-oldloc.v)>15) or (abs(newloc.h-oldloc.h) >15))
         then
           begin
           nbp:=nbp+1;
           ptab[nbp]:=oldloc
         end;
           oldloc:=newloc
```

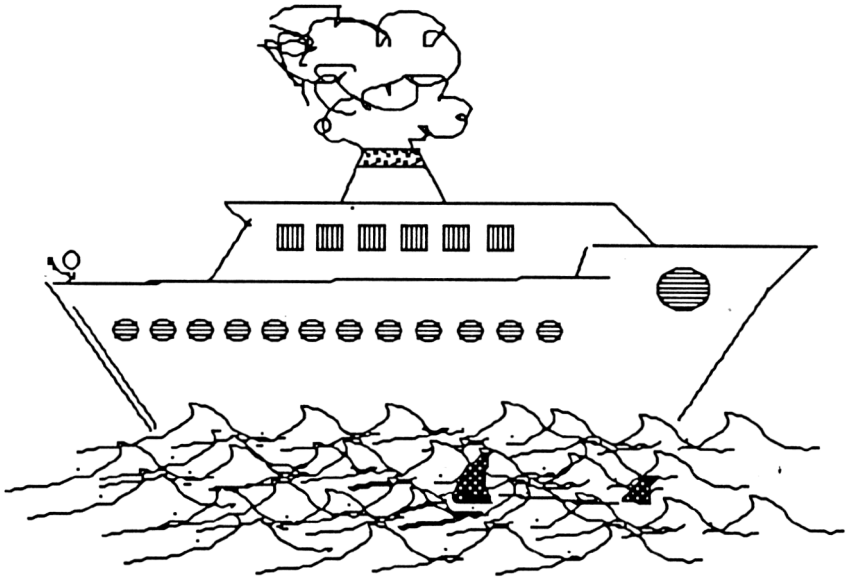
```
        end;
      until not Button;
      go:=false;
      end;

    until doneflag;

    { L'utilisateur a choisi Quit dans le menu File }
    { qu'on ne voit pas }

    endrgn;

    { démo de picture Quickdraw }

    useQDpic;
    end; (* mainloop *)


{ programme }
  begin
  initall;
  showpict;
  DrawMenuBar;
   UnloadSeg(@SetupMenus);
  mainloop;
  end.
```

Ce style de programmation est assez similaire à celui du "C".

A présent, voyons les éléments qui nous manquent pour étoffer un peu nos connaissances.

CHAPITRE XIII

PRINT, SCRAP, E/S, SON, PERIPHERIQUES, COMMUNICATIONS.



Ce dessin n'a vraiment, mais alors vraiment rien à voir avec le sujet. On aurait très bien pu dessiner ici un camion pour y entreposer la somme de connaissances qui restent à aborder.

Mais nous ne verrons pas tout: nous nous cantonnons aux principes de base; ultérieurement, et avec le compilateur de votre choix (Megamax "C" avec RamDisk, et un autre éditeur que celui de Megamax?), vous vous mettrez si le coeur vous en dit aux expériences personnelles. Pour l'instant, revenons au Print Manager.

PRINT MANAGER

Le Print Manager appellera des routines identiques, peu importe l'imprimante qui est connectée. Le 'DRVR' (ici, le sens en est 'printer driver', pilote de l'imprimante) est contenu dans une des ressources du système. Si vous connectez d'autres imprimantes, vous travaillez avec un autre driver, mais continuez d'effectuer les mêmes appels aux mêmes routines d'impression.

Imprimer à partir d'un Mac, c'est tout à fait comme afficher à l'écran. On peut imprimer en mode "brouillon" ou en écrivant sur un fichier d'impression, en déclarant *port courant* le port de l'imprimante (TPPrPort). Alors, le fichier d'impression est enregistré juste avant l'impression. Ensuite, le fichier enregistré est envoyé ligne par ligne à l'imprimante.

Pour l'ImageWriter, les lignes sont des **bandes de graphisme**.

Ce processus est très gourmand en mémoire. On place en général tout ce qui est relatif à l'impression dans un même segment, et on l'exécute après avoir purgé tous les autres segments de code appelés précédemment pour d'autres tâches. Ensuite, on purge ce segment.

Le Print Manager offre les dialogues standard pour l'impression et la Mise en page.

Avant toute opération, il faut ouvrir le port d'impression au moyen d'**OpenPPort**. Ensuite, on définit un record qui contiendra toute l'information relative à l'impression. Ce, afin de conserver les valeurs fixées (Lettre américaine, ...).

PrintDefault donne au fichier d'impression un tel record avec des valeurs par défaut. **PrValidate** vérifiera la structure et la cohérence du record avant de l'utiliser.

PrStlDialog et **PrJobDialog** permettent à l'utilisateur de modifier les paramètres. On récupère alors les valeurs de manière cohérente. Voyez la syntaxe de ces routines dans l'Appendice.

Quelques mots au sujet du texte en brouillon (draft mode): il s'imprime très facilement en utilisant les routines **PRDRVROpen** en début de programme, **PRCTLCall** qui imprime toute fenêtre ou tout écran, et **PRDRVRClose** en fin de programme. Ces routines appellent directement le Print Driver.

Hors du draft mode, on doit créer un document. **PrOpenDoc** l'ouvrira. **PrOpenPage** cadrera chaque page, et **PrClosePage** fermera le document. Une ligne étant une chaîne de graphisme, on imprime chaque ligne par **PrPicFile**. On ferme le document en utilisant **PrCloseDoc**. Toutes les routines ne retournent pas de code d'erreur, mais en général **PrErr** vous dira si tout se passe bien, et sinon, ce qui ne va pas, selon le code.

En fin d'impression, on ferme le port d'impression d'un **PrClose**.

LE MYSTERIEUX PRESSE-PAPIERS

Nous avons déjà souligné le fait qu'il existe deux presse-papiers (clipboard): un est spécifique à l'application, l'autre est un moyen de passer des données entre applications. C'est le second que nous abordons à présent.

Le Segment Loader charge en memoire les applications et gère les segments.
En lançant une application, il transfère le ClipBoard, d'abord situé dans le heap du système, vers le stack. Le heap est ensuite initialisé pour recevoir une nouvelle application, et ensuite, le contenu du ClipBoard est recopié dans le heap. A ce moment, on

a donc **deux** ClipBoard.

En fin d'application, il faut transférer le "scrap local" (rappelez-vous TEScrap,...) vers le scrap "public", si le scrap de l'application n'est pas vide.

Deux types standard sont reconnus par le ClipBoard: **TEXT**, et **PICT**. Le type PICT consiste en une série d'instructions qui dessinent quelque chose, et là, peu importe quoi. On peut définir ses propres types. Le ClipBoard ne peut contenir qu'un bloc, et une seule catégorie d'informations à un instant donné.

Avant d'utiliser le ClipBoard, une application doit savoir la taille de celui-ci, afin de déterminer s'il y a assez de mémoire pour contenir le programme, et le contenu du ClipBoard. Sinon, celui-ci sera partiellement copié sur le disque. Pour économiser de la place.

La mémoire occupée par un programme est celle du segment principal, des segments actifs, des ressources chargées et, éventuellement, du ClipBoard.

SCRAP MANAGER

La ToolBox a une définition d'un type spécial, **ResType**, qui définit le type de scrap:

```
type
    ResType = packed array[1..4] of char;
```

Le format général du ClipBoard est le suivant:

4 bytes : type de données (ResType).
4 bytes : longueur de la donnée (en octets).
n bytes : données. Nombre pair d'octets, sinon, padding.

Appelez **InfoScrap** pour connaître l'espace mémoire occupé par le ClipBoard. Ici encore, la ToolBox nous fournit une structure. La voici:

```
type
    PScrapStuff  =   ^ScrapStuff;
    ScrapStuff   = record
                       scrapSize    :   LongInt;
                       scrapHandle  :   Handle;
                       scrapCount   :   Integer;
                       scrapState   :   Integer;
                       scrapName    :   StringPtr;
                   end;
```

Voici comment on utilise cette routine:

function InfoScrap: PScrapStuff;

- scrapSize contient la taille du ClipBoard.

- scrapHandle pointe sur l'information contenue dans le ClipBoard, s'il y en a, ou NIL (Ø, en "C"), si le ClipBoard n'en contient pas.

- scrapCount indique si le contenu du ClipBoard a été modifié en utilisant un accessoire de bureau.

- si le scrap est en mémoire, scrapState > Ø.
- si le scrap est sur disque, scrapState = Ø.

La procédure **ZeroScrap**, qui remet le scrap à zéro et modifie scrapCount, est appelée avant tout transfert de scrap local vers le scrap "public", le ClipBoard.

GetScrap(Dest: Handle; whichType: ResType
var Offset: LongInt): LongInt;

charge les données d'un certain type (spécifié dans le champ whichType) dans le scrap (en mémoire ou sur disque. GetScrap n'effectuera aucune vérification. Il faut donc être certain d'avoir assez de place en appelant InfoScrap).

GetScrap retourne un code. Si celui-ci est < Ø, tle code signifie pourquoi le code-retour est négative. La constante suivante:

const

noTypeErr = -102;

signifie que le type spécifié dans whichType ne se trouvait pas dans le scrap. Si ce code st positif, il contient la longueur de l'information chargée.

PutScrap(lengt: LongInt; whichType: ResType; src: Ptr): Longint;

La donnée du type spécifié dans whichType, de longueur "lengt", est accédée via le pointeur **src**. Elle est chargée dans le ClipBoard. Avant d'appeler cette routine, il faudra retire du scrap les données du même type en appelant ZeroScrap.

Le résultat est égal à Ø si aucune erreur ne s'est produite.

Nous en avons terminé avec le Scrap Manager. Maintenant, dirigeons-nous vers les entrées/sorties.

FILE MANAGER

Le but du File Manager est de transférer un fichier du Mac vers un périphérique, et vice-versa. Un fichier est une ensemble de données, souvent de même structure, stocké sur une mémoire de masse. Un fichier a un nom, la mémoire de masse possède un numéro de volume. Les routines du File Manager servent à effectuer toutes les opérations de lecture et écriture des fichiers.

Sur le Mac, le nom de volume peut avoir jusqu'à 27 caractères. Le volume contient une directory de ses fichiers (un index général). Cette directory contient les adresses des blocs de début de chaque fichier (la longueur d'un bloc est de 512 octets). Un volume peut représenter une unité complète, ou partielle, selon le type de mémoire de masse (disquette ou disque dur).

Le numéro d'avril 1985 (n° 5) de MacTutor vous offre une excellente explication de ce qu'est une directory vue par Mac. C'est pourquoi nous ne vous l'offrons pas ici.

Un volume peut être soit monté (mounted), soit non-monté (unmounted). Monté, il occupe 1,5 Koctets. Non monté, son icone apparaît en grisé. Il n'occupe alors que 94 octets.
Le système assigne un n° de référence (utilisé dans les routines) à tout volume monté.

Sur Mac, les fichiers peuvent comprendre jusqu'à 16 millions d'octets, et on leur attribue un nom (jusqu'à 64 caractères). Ils ont aussi un numéro de version. Un fichier a aussi un **type** (un ResType), utilisé par le Finder.

Ce type et le créateur qui l'accompagne servent au Finder pour ouvrir l'application qui l'a généré.

Un fichier Mac est composé de deux sections: la "fourchette" de ressources, et la fourchette de données. La première peut être vide, mais sert à localiser le fichier (comme certaines chaînes de caractères, mais encore des formats de monnaie - pour une chaîne de comptabilité, par exemple).

Un fichier est défini par son premier octet, n° Ø. La fin de fichier **logique** se trouve un octet après le dernier octet du fichier. La fin de fichier physique se trouve un octet après le dernier bloc utilisé par ce fichier.

Le Standard File (SF) Package vous permet d'ouvrir, lire, écrire, fermer des fichiers, ainsi que certaines routines du File Manager.

SFGetFile sert à choisir un fichier dans une directory. On crée le document et on vérifie qu'il n'existe pas déjà en utilisant le code retour fourni par **SFPutFile**. Ces deux routines sont accompagnées par les routines GetFInfo, et SetFInfo, qui servent à modifier ou non les paramètres (comme la position sur l'écran, et le classement dans un folder, pardon, dossier, par exemple).

Ce qui complique un peu l'utilisation des fichiers sous Mac, c'est la présence de deux systèmes différents, MFS et HFS (**H**, pour Hierarchical), de stockage et récupération de fichiers. Et aussi, la possibilté d'utilisation de plus d'une routine pour ouvrir, créer, fermer, lire, écrire des fichiers. L'exemple précédent d'utilisation du Pascal TML montre l'utilisation des routines **hautes** d'ouverture et d'écriture d'un fichier MFS.

Vous pouvez combiner l'appel à une routine avec un

traitement éventuel d'erreur en écrivant une routine qui accepte comme paramètre la valeur du code retour d'une des routines d'un de ces managers:

```
procedure MyErrProc(Err: integer);
begin
    IF Err <> Ø THEN
    { ou case.. of }
        ............
    END;
end;
```

Dans le corps du programme, vous utiliseriez cette procédure comme suit:

MyErrProc(**FSOpen**(MyFile, VolRef, FileRef));

qui effectuera ce qu'il y a lieu de faire, dépendemment du code retourné par la fonction.

Après FSOpen, **FSRead**, **FSWrite** and **FSClose** permettent respectivement de lire, écrire et fermer un fichier (routines hautes). Ils ont déjà été utilisés dans les exemples, donc, nous vous renvoyons à ceux-ci.

La fermeture d'un fichier écrit le dernier bloc, et libère le buffer qu'il occupait.

Très important, **FlushVol** remet la directory à jour. On l'utilise en général après un "close". Comme, par exemple, chaque fois que vous sauvegarderez le fichier index d'un séquentiel indexé.
Nous trouvons encore des routines telle **Eject**, qui effectue elle aussi un FlushVol, **GetVInfo** (information au sujet d'un volume. Un des paramètres passés à cette routine, c'est le n° de lecteur: -1 sera le startUp disk, 2, le lecteur externe, et 3 sera le RamDisk), **GetVol** (fournit les caractéristiques du volume par défaut), **SetInfo** (fixe le volume par défaut).

On peut aussi accéder directement à un octet particulier dans un fichier en se servant de **GetFPos** et **SetFPos**.

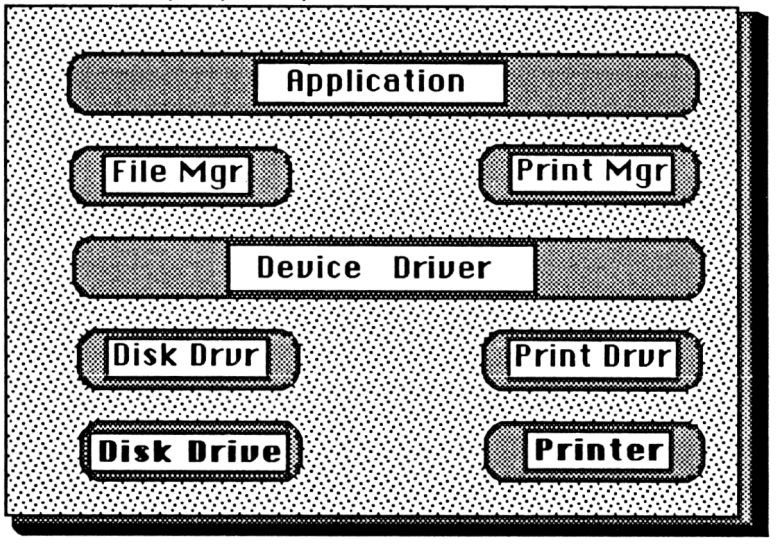
Voici à présent quelques mots au sujet de l'interface entre les applmications et le monde extérieur: le Device Manager.

DEVICE MANAGER

Les périphériques sont soit internes au Mac (lecteur de disquette, générateur de son), soit externes (imprimantes, disque dur). Les différences entre périphériques se retrouvent au niveau du transfert des données, et au type de stockage.

Le périphérique peut transférer ou recevoir de l'information caractère par caractère, ou bloc par bloc. Dans un transfert par bloc (lecteurs, disques durs), les blocs sont transférés dans un sens et dans l'autre **directement** à leur adresse (FBA).

La Fig. suivante montre la couche de logiciel située entre les applications et le périphérique.:



L'Operating System comprend trois drivers en ROM: le Sound Driver, le disk driver, et l'I/O driver.

Le printer driver est situé en RAM. D'autres drivers gèrent les accessoires de bureau. La tâche du driver est de transférer les

données depuis et vers une application. Un buffer de transit sera créé, accédé par un pointeur.

Deux drivers, de son et de disque, sont constamment ouverts par le système. Celui-ci ne peut gérer que 32 drivers ouverts simultanément.

On se sert d'un driver par son numéro, compris entre -1 et -32. C'est la seule identification qui sera utilisée. Le nom d'un driver (DRVR) en RAM est le même que celui du fichier ressources.

Les routines de haut niveau d'accès aux Drivers sont les suivantes:

OpenDriver pour ceux situés en RAM.

Control et **Status** seront respectivement utilisés pour gérer le driver, et connaître l'état de ceui-ci.

KillIO vide le contenu du buffer d'E/S (I/O buffer), et **Close** termine l'E/S en cours.

Une routine gérant un driver peut travailler en mode synchrone. Le programme, s'il est interrompu, se poursuivra jusqu'à ce que la tâche I/O soit terminée. En mode asynchrone, un programme continue de s'exécuter pendant que l'E/S s'effectue.

On peut écrire ses propres drivers, mais nous sauterons cette partie, ne nous occupant que de ce qui existe.

COMMUNICATIONS

On trouve deux ports séries sur le Mac. Le port A - modem -, et le port B - imprimante série. Sur le Mac +, on trouve aussi le portSCSI, à haute vitesse de transfert. Chacun a son driver d'entrée, et son driver de sortie, en ROM.

Une version RAM accepte le protocole Xon/Xoff protocol. Tous

supportent les communications asynchrones, à des vitesses allant de 300 à 57600 bauds. Le contrôleur de communications série est un Zilog 8530.

La valeur prise par défaut par les drivers est de 9600 bauds.

Une communication série est constituée d'un certain nombre d'octets à transmettre. Chaque octet est accompagné d'un bit de départ, suivi des 7 ou 8 bits à transmettre, accompagnés d'un bit de parité, et finalement, le(s) bit(s) d'arrêt.

Le **frame** (cadre), c'est l'intervalle de temps qui sépare le bit de départ du dernier bit d'arrêt.

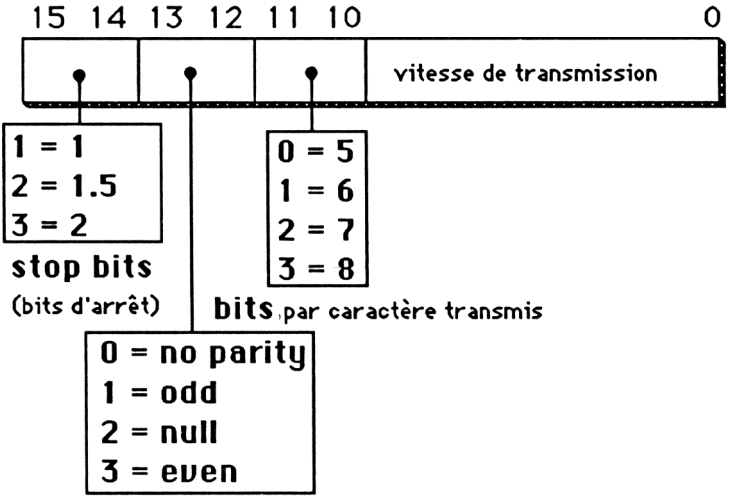
Pour ouvrir ces drivers, on trouve quelques routines:
Open sert à ouvrir les drivers en ROM. Les drivers en RAM doivent être ouverts par **RAMDOpen**. Il ne faut pas fermer un driver en ROM (**Close**) sans ouvrir simultanément le driver en RAM: la souris est partiellement contrôllée par le SCC Z8530, et il vous sera impossible (le pointeur de la souris est en RAM) de gérer les mouse events jusqu'au prochain "boot".

Pour femer un driver en RAM, vous appelerez **RAMDClose**. Les deux types de drivers (ROM et RAM) utiliseront **FSRead** et **FSWrite**, pour transférer les données depuis et vers un périphérique.

Naturellement, on dispose de fonctions spécifiques pour fixer les paramètres de transmission:

SerReset(RefNum, Config);

impose au driver spécifié par son numéro (RefNum) les valeurs placées dans Config; vous trouverez ci-dessous la structure de l'entier Config:



SerHShake(RefNum, HFlags) permet le contrôle d'un handshake. HFlags est un record. Ses champs permettent ou annulent les modes CTS ou Xon/Xoff; quand à ce dernier, ils définissent aussi les caractères qui l'activent.

On utilise le "frame" comme unité de temps. Si la ligne est libre pendant plus d'un frame, le système récépteur prend cette information comme une coupure (break). **SerSetBrk** et **SerClrBrk** activent et désactivent respectivement le mode "break".

D'autres routines permettent de définir ou lire un enregistrement contenant des erreurs de transmission.

La table ci-dessous montre les correspondances existant entre les DRVR et les numéros de référence:

Driver de:	**Nom**	**N° réf.**
modem input (A)	.AIn	-6
modem output (A)	.Aout	-7
printer input (B)	.BIn	-8
printer output (B)	.Bout	-9

Notre tour des drivers est terminé. Maintenant, nous passons au son (!).

SON

Programmer un Mac pour produire de la musique est amusant, mais pas extrêmement facile. Il nous est arrivé de connecter un Mac à une chaîne HI-FI, et le résultat fut indicible.

On peut considérer le Mac comme un pré-amplificateur. Il suffit de connecter la sortie audio sur *AUX IN* de la chaîne pour entendre la différence. En fait, ce n'est pas **UN** mais trois synthétiseurs, que l'on trouve dans le Mac.

L'un génére des signaux carrés, un autre, une sinusoïde, et le troisième est un "free form synthesizer" (synthétiseur libre).
Un signal est composé d'une fréquence (la note), une amplitude, et la phase. Tout se trouve placé dans un buffer sous forme digitale. Les deux premiers octets sélectionnent le type de synthétiseur, et la couche du record associé.

Signal carré

Un signal carré oscille entre deux niceaux à une fréquence constante. Vous trouvez ci-dessous la structure du record associé à un signal carré:

```
type
    Tone = record
              count        : integer; { fréquence }
              amplitude    : integer; { amplitude. 0..255 }
              duration     : integer; { 0..255 ticks }
           end;

    Tones: array[0..5000] of Tone;

    SWSynthPtr  = ^SWSynthRec;

    SWSynthRec =
        record
```

```
        mode     : integer; (* ici, -1pour le signal carré *)
        triplets : Tones;
    end;
```

Vous pouvez générer jusqu'à 5000 sons successifs. Le dernier élément de la table doit contenir Ø.

On obtient la fréquence par la formule suivante:783360 / count (en Hertz).

Ne définissez pas de variable de type Tone. Créez plutôt un bloc relogeable contenant les données relatives au son, ensuite, appelez **StartSound**. Vous dé-verrouillerez le handle pointant vers la structure après usage, et après, pfuit!, DisposeHandle.

Pour créer de la vraie musique, rien de tel qu'un record FTSynthRec. La fréquence ira de Ø to 22.3 KHz.

Pour créer les signaux "libres", proches de la voix humaine, on utilisera FFSynthRec. Le Pascal TML fournit lui aussi une possibilité de transformer des chaînes de caractères en voix parlée (MacinTalk), et MacTutor donne des exemples en ce langage.

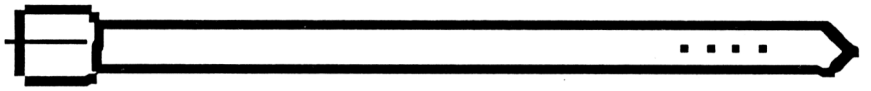
On dispose encore d'autres routines, comme par exemple **SoundDone**, **StopSound**, **SetSoundVol** et **GetSoundVol**.

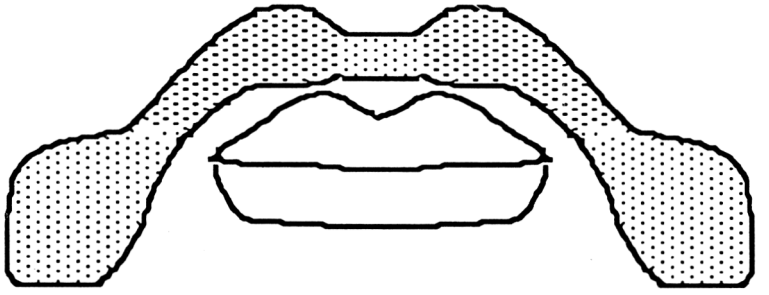
Le prochain chapitre vous fournit d'autres détails afin de vous lancer dans le développement, ainsi que l'étude de certains utilitaires.

CHAPITRE XIV

LE COIN DES DEVELOPPEURS

ET DES UTILISATEURS





FICELLES

Ce dessin n'est pas censé représenter des ficelles, mais une ceinture et des favoris. Qui se traduisent par "belts and whiskers", très proche, phoniquement de "bells and whistles", l'équivalent de "ficelles". Clair?

Ne nous égarons pas...

Nous présentons à présent d'autres solutions, et quelques

morceaux choisis de code. Nous avons proposé une autre solution aux update events. La voici:

```
{ cet exemple stocke le contenu d'une fenêtre dans un bitimage }

  var
      myWindow          :    WindowPtr;
      srcRect, dstRect  :    Rect;
      tempBits          :    bitmap;
      sz                :    size;
      P                 :    Ptr;
      OldPort           :    Grafptr;
```

La sauvegarde du contenu sera effectuée par:

```
  ..........................................
  Getport(OldPort);
  SetPort(myWindow);   { fixe le port courant }
  srcRect:= myWindow^.portRect; { stocke portRect dans
srcRect }
  dstRect:=srcRect;        { deux rectangles égaux }
  tempBits.rowBytes:= myWindow^.portBits.rowBytes;
{ sauvegarde les coordonnées de la fenêtre active }

  tempBits.bounds:= myWindow^.portBits.bounds;
{ save limits of active bitmap }
  sz:=   ORD4(tempBits.rowBytes*(tempBits.bounds.bottom
tempBits.bounds.top));
{ calcule la taille de l'image }
  P:= NewPtr(sz);
{ pointeur + place non-relogeable de taille égale à sz }
  tempBits.baseAddr:= POINTER(P);
{ P pointera sur ce bitimage }
  CopyBits (myWindow^.portBits, tempBits, srcRect, dstRect, srcCopy, NIL);
{ Ouf, le voilà sauvegardé! }
```

Lors de la mise à jour:

```
  CopyBits (tempBits, myWindow^.portBits, dstRect, srcRect, srcCopy, NIL);
{ On copie à l'envers. Notez l'inversion des paramètres }
  DisposPtr(P);
  { on respire }
  SetPort(OldPort);
  { on restaure }
```

Méthode rapide, qui fonctionne bien. Voulez-vous rebooter sans repasser par la case "Finder"? Rien de mieux qu'un reset du système par une routine en assembleur:

```
.PROC MyRest
MOVE.L    #$40000A,A1      ; l'adresse clé!
JMP       (A1)
.END
```

Voici une fonction très courte, que vous pouvez utiliser pour trouver une fenêtre particulière. Elle vous rend le WindowPtr que vous cherchez:

```
function kindWindow(StartWind: WindowPtr;
                    kind: integer): WindowPtr;

var
    thePeek: WindowPeek;
begin
    thePeek:= WindowPeek(StartWind); { pointe sur la }
            { fenêtre de début }

    kindWindow:= NIL;
    WHILE thePeek <> NIL DO
    begin
        IF (thePeek^.Windowkind = kind) THEN
        begin
            kindWindow:= WindowPtr(thePeek);
            EXIT(kindWindow);     { window found }
        end       { if }
        ELSE
            thePeek:= thePeek^.NextWindow; { la suivante }
    { revoyez la structure du window record }
    end;{ while }
end;{ kindWindow }
```

OU OBTENIR DES RENSEIGNEMENTS?

Si vous ne développez pas encore de logiciels et que vous voulez transformer ce loisir en gagne-pain, téléphonez à Apple France, pour connaître les formalités nécessaires afin d'être

enregistré: vous recevrez une certaine quantité d'informations, dans la Lettre des Développeurs. Ou encore, faites appel à un développeur pour réaliser une idée. Apple France peut placer une annonce pour vous. Ces suggestions n'ont rien d'incongru: nous connaissons des dentistes et des médecins, pas en panne de patients, qui, un beau jour, se sont transformés en développeurs. Et avec succès.

Si vous voulez des tuyaux logiciels, rien de mieux que MacTutor, que vous trouverez dans certaines boutiques.

MacMag (Montréal) avait de bonnes rubriques sur l'assembleur. Mais on ne le voit plus souvent.

MacWorld, Macazine, MacUser, le Macintosh Buyer's guide, et A+, ce sont les sources ultimes d'information. Ainsi que l'inévitable Byte. Sans oublier les réseaux américains, auxquels on peut accéder de France: Compuserve et Delphi.

Quant au réseau Calvados, il vous met en contact avec les développeurs français.

Nous avons mentionné **Inside Macintosh**™ comme la lecture de chevet du développeur. Il est commercialisé en version brochée (Addison Wesley) aux alentours de 600 Francs. C'est une mine d'informations aride, sans exemples, dans lequel vous trouverez cependant une table des codes ASCII et hexadécimaux que nous n'avons pas inclus. Cet ouvrage est indispensable au développeur.

Après **FS**reading ce livre, vous comprendrez Inside Macintosh.

Si vous vous sentez l'âme d'un organisateur, créez un club, sinon, adhérerez au Club MAK: c'est le seul à notre connaissance qui réunisse des utilisateurs et des développeurs.

Si vous voulez programmer, servez-vous du Basic, du "C" (nous sommes résolument pour le Megamax) ou encore du Pascal TML. Sans oublier l'inévitable RAM Disk sans lequel vous passeriez des heures devant un Mac compilant.

Autres applications, autres langages: l'intelligence artificielle s'accorde bien de langages comme SmallTalk, Lisp, ou Prolog. Le Forth séduit certains. Et même le Basic compilé (Pterodactyl),

ou encore NEON. Les microprocesseurs sont plus rapides, les langages de haut niveau fleurissent...

Deux développeurs français se sont penchés sur les compilateurs Modula II. L'un (GEFI Services a déjà sorti un produit. Nous ne l'avons pas encore testé. Pas mal, Modula II. Le second travaille seul. Nous lui souhaitons tout le succès que son travail en solitaire mérite: Pascal Bourguignon. Il proposera certainement le fruit de son travail aux Etats-Unis.

DEBUGGING

Le Debugging c'est la possibilité de découvrir pourquoi un programme (à ce niveau, ce n'est encore un programme. Ce n'est qu'après moult tests, un conditionnement - packaging - et un manuel d'utilisation lisible qu'il atteint le niveau de logiciel) ne fonctionne pas comme celui qui l'a écrit le désire.

C'est aussi savoir à quel endroit dans le programme quelque chose ne va pas. MacsBug, sans oublier le papillon de "break" sur le côté gauche - en le regardant - du Mac, c'est le débugger du Mac.

Au chargement du système, un message "MacsBug installé" apparaît. Vous êtes rassuré. Et vous jouerez avec la profusion d'ordres DM, ES, MR, HD, IL (calmons-nous) pour réparer le code déficient.

Mais avant de débugger, il faut savoir construire un programme. D'abord, savoir utiliser l'utilitaire Dialog Creator, qui vous permet de construire des dialogues qont les zones ne se recouvrent pas. Car deux zones se chevauchant peuvent "geler" une fenêtre, sur le Mac.

Ensuite, il faut commencer par écrire de petits programmes. Et savoir quels modules, dans une bibliothèque que vous avez créée et testée, utiliser quand vous passez aux programmes plus importants.

AUTRES OUTILS: LE RESOURCE EDITOR, SETFILE

Si vous vous branchez sur certains des réseaux cités précédemment, ou si vous vous procurez le Mac Software Supplement (de chez Apple), ou encore, si vous avez des amis qui vous les prêtent, essayez votre science de Mac avec le **Resource Editor**, et **SetFile**. Ce, afin d'entrer dans les ressources.

Vous pourrez faire une liste des raccourcis qu'utilisent vos logiciels favoris, en rajouter, en bref, les **personnaliser**. Nous utilisons souvent l'option "Insérer un saut de page" de MacWrite, item de menu pour lequel il n'existe pas de raccourci. Après avoir examiné MacWrite et constaté que la lettre "E" n'est pas utilisée comme raccourci, nous avons décidé d'utiliser l'ensemble "Commande-E" comme raccourci de "Insérer un saut de page".

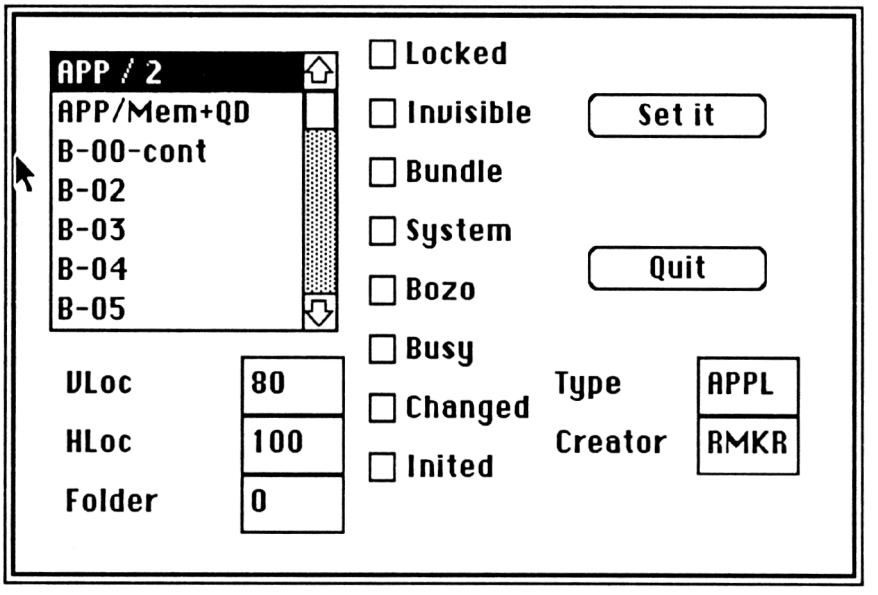
Le Resource Editor permet aussi de modifier les chaînes de caractères situées dans les types de ressources 'STR '. Vous pouvez à volonté modifier les messages qui apparaissent dans vos logiciels favoris. Ou encore, les Menus, la taille des fenêtres et des dialogues. Mais faites une copie de sauvegarde du logiciel à modifier auparavant. Et gardez-la en lieu sûr!

Vous pouvez aussi dupliquer et éditer des Fonts, créer vos caractères personnels, et même votre signature.

Certaines ressources, de type INIT, contiennent des messages encryptés, comme "Bienvenue" qui s'affiche lors du boot. Vous pourrez les transformer en "C'est moi le meilleur", ou autre message d'encouragement.

Le Finder affiche les icones d'application et de document spécifiés dans les BNDL et FREF. Vous pourrez modifier ces icones. Toulours avec le Resource Editor. Et aussi, persuader votre Mac que les documents créés par une application peuvent servir à une autre: un document produit par OverView peut très

bien avec le type et le créateur spécifiques à Multiplan, être récupéré par cette application. Vous obtiendrez donc des données immédiatement digestibles par cette application, sans aucun **Couper** ni **Coller**. SetFile, un autre utilitaire, plus simple d'emploi, peut vous aider pour la circonstance.



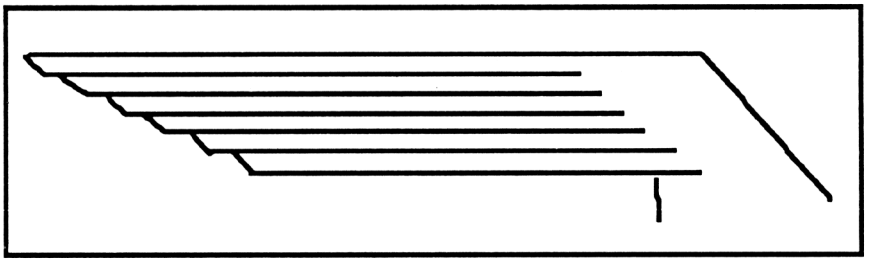
On trouve encore d'autres utilitaires, tel 'Fedit', 'MacZap', et 'Examine File'. Testez-les sur des fichiers dupliqués, toujours pour des raisons de sécurité.`

Voici le moment de l'au revoir. Nous espérons quer vous aurez apprécié ce livre, et sinon, qu'il vous aura été utile pour comprendre l'intérier de cette merveilleuse machine, le Macintosh. Nous n'avons pu tout couvrir en ces lignes, et savons fort bien que des sujets importants ont été éclipsés. Par contre, nous avons mis l'effort sur tout ce qui est fenêtres, dialogues, souris et événements.

Mais si, après les exemples commentés vous vous sentez confiants, si vous avez souri en constatant que tout cela n'est pas si compliqué, après tout, si vous savez parfaitement ce que sont les handle, ou encore Fenêtres et Menus, et les types de ressources correspondants, alors nous aurons réussi notre

tâche.

Et nous vous remercions de votre patience...



APPENDICE A [adresses]

LANGAGES (En $)	COMPAGNIE	PRIX
MacAdvantage	SofTech	295
UCSD- Development system	SofTech	195

SofTech Microsystems, Inc.;
16875 West Bernardo Dr.; San Diego, Calif. 92127;
800-451-8080 ou 800-824-7867 en Calif.

MacAsm	Mainstay	125

Mainstay;
28611B Canwood St.; Agoura Hills, Calif. 91301;
818-991-6540

Mac C et Mac C Toolkit	Consulair	375

Consulair Corporation;
140 Campo Dr.; Portola Valley, Calif. 94025;
415-322-2757

MacForth	Creative Solutions	
niveau 1		149
niveau 2		249
niveau 3		499

Creative Solutions, Inc.;
4701 Randolph Rd., Suite 12, Rockville, Md. 20852;
301-984-0262.

MacFortran	Absoft	395

Absoft Corp.;
4268 N Woodward; Royal Oak, Mich. 48072;
313-549-7111

Macintosh Pascal	Apple Computer	125

Apple Computer, Inc.;
20525 Mariani Ave.; Cupertino, Calif. 95014;
408-973-3317

LANGAGES (En $)	COMPAGNIE	PRIX
Mac-Modula II	Modula Corporation	150

Modula Corporation;
950 N University Ave.; Provo, Utah 84604;
800-545-4842

Megamax C	Megamax	299.95

Megamax, Inc.;
PO Box 851521; Richardson, Texas 75085;
214-987-4931

Microsoft BASIC	Microsoft	150

Microsoft Corporation;
10700 Northup Way, PO Box 97200; Bellevue, Wash. 98009;
206-828-8080

Neon	Kriya	150

Kriya Systems, Inc.;
505 N Lake Shore Dr., #5510; Chicago, Illinois 60611;
312-822-0624

Softworks C	Softworks	395

Softworks Limited;
607 W Wellington; Chicago, Illinois 600657;
312-975-4030

RESEAUX

Compuserve;
5000 Arlington Boulevard; Colombus, Ohio 43220;

APPENDICE B

[MEMTYPES, MEMORY MGR]

Les routines sont précédées d'un "**P**" si ce sont des procédures, par un "**F**" si ce sont des fonctions. L'**INLINE** correspondante est signalée par le n° d'**INLINE**, débutant par un "**A**", et ne sont précédées par aucun symbole (comme un '$').

MEMTYPES

```
type
  SignedByte=-128..127;          { Nous avons déjà }
  Byte=0..255;                   { vu ces types en étudiant }
  Ptr=^SignedByte;               { le Memory Manager }
  Handle=^Ptr;
  ProcPtr=Ptr;
  Fixed=longint;                 { pour virg. fixe }
  Str255=String[255];            { longueur de chaîne max. }
  StringPtr=^Str255;             { pointeur sur chaîne max. }
  StringHandle=^StringPtr;       { handle sur chaîne max. }
```

MEMORY MANAGER / SEGMENT LOADER

```
P HLock(h:Handle);
P HUnLock(h:Handle);
P HPurge(h:Handle);              { marque le bloc purgeable }
P HNoPurge(h:Handle);
P MoreMasters;                   { au début d'un programme }
                                 { avant MaxApplZone, pour }
                                 { allouer un autre bloc de 64 }
                                 { mps; use it }
                                 { avant toute allocation de }
                                 { mémoire }
F NewPtr(byteCount:Size):Ptr;
P DisposPtr(p:Ptr);
F GetPtrSize(p:Ptr):Size;
P SetPtrSize(p:Ptr;newSize:Size);
F NewHandle(byteCount:Size):Handle; {s'il retourne NIL, }
{ testez MemError }
F MemError:OsErr; { code retour du dernier appel au MemMgr }
P DisposHandle(h:Handle);
F GetHandleSize(h:Handle):Size; { pour obtenir la taille logique }
                                { du bloc associé au handle }
P SetHandleSize(h:Handle;newSize:Size); { Pour fixer cette }
```

```
{taille. }
F  CompactMem(cbNeeded:Size):Size; { ne purge pas }
P PurgeMem(cbNeeded:Size);
                        { purge les blocs, ne compacte pas }
P ResrvMem(cbNeeded:Size);
F  FreeMem:longint;                  { Obtenir la taille mémoire }
{ libre disponible dans le heap }
F  MaxMem(var grow:Size):Size; {purge les purgeables }
F  TopMem:Ptr;
P SetGrowZone(growZone:ProcPtr);
P SetApplLimit(zoneLimit:Ptr);
F  PtrZone(p:Ptr):THz;
F  HandleZone(h:Handle):THz;
F  GetApplLimit:Ptr;
P MaxApplZone;
F  MoveHHi (h:handle):OsErr;
P EmptyHandle(h:Handle);
P ReAllocHandle(h:Handle;byteCount:Size);
P BlockMove(srcPtr, destPtr:Ptr;byteCount:Size);
F  GZCritical:boolean;
F  GZSaveHnd:Handle;
P SetApplBase(startPtr:Ptr);
P InitApplZone;
P InitZone(pgrowZone:ProcPtr;cmoreMasters:integer;
                                    limitPtr,startPtr:Ptr);
F  GetZone:THz;
P SetZone(hz:THz);
F  ApplicZone:THz;
F  SystemZone:THz;
```

Segment Loader

```
P UnLoadSeg(routineAddr:Ptr);   A9F1
P ExitToShell; A9F4;{ relance le Finder }
P GetAppParms(var apName:str255;var apRefNum:integer;
                                    var apParam:Handle); A9F5;
P CountAppFiles(var message:integer;var count:integer);
P GetAppFiles(index:integer;var theFile:AppFile);
P ClrAppFiles(index:integer);
```

APPENDICE C [QUICKDRAW]

GrafPort

```
P InitGraf   (globalPtr: Ptr);                          A86E;
P OpenPort   (port: GrafPtr);                           A86F;
P InitPort   (port: GrafPtr);                           A86D;
P ClosePort  (port: GrafPtr);                           A87D;
P SetPort    (port: GrafPtr);                           A873;
P GetPort    (var port: GrafPtr);                       A874;
P GrafDevice (device: integer);                         A872;
P SetPortBits(bm: BitMap);                              A875;
P PortSize   (width,height: integer);                   A876;
P MovePortTo (leftGlobal,topGlobal: integer);           A877;
P SetOrigin  (h,v: integer);                            A878;
P SetClip    (rgn: RgnHandle);                          A879;
P GetClip    (rgn: RgnHandle);                          A87A;
P ClipRect   (r: Rect);                                 A87B;
P BackPat    (pat: Pattern);                            A87C;
```

Routines de curseur

```
P InitCursor;                                           A850;
P SetCursor(crsr: Cursor);                              A851;
P HideCursor;                                           A852;
P ShowCursor;                                           A853;
P ObscureCursor;                                        A856;
```

Routines de tracés lignes

```
P HidePen;                                              A896;
P ShowPen;                                              A897;
P GetPen     (var pt: Point);                           A89A;
P GetPenState(var pnState: PenState);                   A898;
P SetPenState(pnState: PenState);                       A899;
P PenSize    (width,height: integer);                   A89B;
P PenMode    (mode: integer);                           A89C;
P PenPat     (pat: Pattern);                            A89D;
P PenNormal;                                            A89E;
P MoveTo (h,v: integer);                                A893;
P Move   (dh,dv: integer);                              A894;
P LineTo     (h,v: integer);                            A891;
P Line(dh,dv: integer);                                 A892;
```

Routines de texte

```
P TextFont    (font: integer);          A887;
P TextFace    (face: Style);            A888;
P TextMode    (mode: integer);          A889;
P TextSize    (size: integer);          A88A;
P SpaceExtra  (extra: Fixed);           A88E;
P DrawChar    (ch: char);               A883;
P DrawString  (s: Str255);              A884;
P DrawText    (textBuf: Ptr; firstByte,byteCount: integer); A885;
F CharWidth   (ch: char): integer;      A88D;
F StringWidth (s: Str255): integer;     A88C;
F TextWidth(textBuf: Ptr; firstByte,byteCount: integer): integer,
A886;
P GetFontInfo (var info: FontInfo);     A88B;
```

Calculs sur des points

```
P AddPt(src: Point; var dst: Point);    A87E;
P SubPt(src: Point; var dst: Point);    A87F;
P SetPt(var pt: Point; h,v: integer);   A880;
F EqualPt     (pt1,pt2: Point): boolean; A881;
P ScalePt (var pt: Point; fromRect,toRect: Rect);   A8F8;
P MapPt (var pt: Point; fromRect,toRect: Rect);     A8F9;
P LocalToGlobal (var pt: Point); A870;
P GlobalToLocal (var pt: Point); A871;
```

Calculs sur des Rectangles

```
P SetRect   (var r: Rect; left,top,right,bottom: integer);   A8A7;
F EqualRect (rect1,rect2: Rect): boolean; A8A6;
F EmptyRect (r: Rect): boolean; A8AE;
P OffsetRect (var r: Rect; dh,dv: integer); A8A8;
P MapRect   (var r: Rect;    fromRect,toRect: Rect); A8FA;
P InsetRect (var r: Rect; dh,dv: integer); A8A9;
F SectRect  (src1,src2: Rect; var dstRect: Rect): boolean; A8AA;
P UnionRect (src1,src2: Rect; var dstRect: Rect); A8AB;
F PtInRect  (pt: Point; r: Rect): boolean; A8AD;
P Pt2Rect   (pt1,pt2: Point; var dstRect: Rect); A8AC;
```

Opérations graphiques sur des Rectangles

```
P FrameRect (r: Rect);                  A8A1;
P PaintRect (r: Rect);                  A8A2;
P EraseRect (r: Rect);                  A8A3;
P InvertRect (r: Rect);                 A8A4;
P FillRect  (r: Rect; pat: Pattern);    A8A5;
```

Rectangles arrondis

```
P FrameRoundRect  (r: Rect; ovWd,ovHt: integer);   A8B0;
P PaintRoundRect  (r: Rect; ovWd,ovHt: integer);   A8B1;
P EraseRoundRect  (r: Rect; ovWd,ovHt: integer);   A8B2;
P InvertRoundRect (r: Rect; ovWd,ovHt: integer);   A8B3;
P FillRoundRect (r: Rect; ovWd,ovHt: integer; pat: Pattern);  A8B4;
```

Opérations sur des Ovales

```
P FrameOval  (r: Rect);                  A8B7;
P PaintOval  (r: Rect);                  A8B8;
P EraseOval  (r: Rect);                  A8B9;
P InvertOval (r: Rect);                  A8BA;
P FillOval   (r: Rect; pat: Pattern);    A8BB;
```

Routines d'arcs

```
P FrameArc  (r: Rect; startAngle,arcAngle: integer);    A8BE;
P PaintArc  (r: Rect; startAngle,arcAngle: integer);    A8BF;
P EraseArc  (r: Rect; startAngle,arcAngle: integer);    A8C0;
P InvertArc (r: Rect; startAngle,arcAngle: integer);    A8C1;
P FillArc   (r: Rect; startAngle,arcAngle: integer; pat: Pattern);
A8C2;
P PtToAngle (r: Rect; pt: Point; var angle: integer);   A8C3;
```

Routines de Polygones

```
F OpenPoly:   PolyHandle;        A8CB;
P ClosePoly;                     A8CC;
P KillPoly    (poly: PolyHandle); A8CD;
P OffsetPoly  (poly: PolyHandle; dh,dv: integer);  A8CE;
P MapPoly     (poly: PolyHandle; fromRect,toRect: Rect); A8FC;
P FramePoly (poly: PolyHandle);  A8C6;
P PaintPoly   (poly: PolyHandle);  A8C7;
P ErasePoly (poly: PolyHandle);  A8C8;
P InvertPoly  (poly: PolyHandle);  A8C9;
P FillPoly (poly: PolyHandle; pat: Pattern); A8CA;
```

Calculs sur des Regions

```
F NewRgn:  RgnHandle;            A8D8;
P DisposeRgn(rgn: RgnHandle);    A8D9;
P CopyRgn (srcRgn,dstRgn: RgnHandle); A8DC;
P SetEmptyRgn(rgn: RgnHandle); A8DD;
P SetRectRgn(rgn: RgnHandle; left,top,right,bottom: integer);
A8DE;
P RectRgn   (rgn: RgnHandle; r: Rect); A8DF;
```

```
P OpenRgn;                                    A8DA;
P CloseRgn (dstRgn: RgnHandle);  A8DB;
P OffsetRgn (rgn: RgnHandle; dh,dv: integer); A8E0;
P MapRgn   (rgn: RgnHandle;   fromRect,toRect: Rect);  A8FB;
P InsetRgn  (rgn: RgnHandle; dh,dv: integer);          A8E1;
P SectRgn   (srcRgnA,srcRgnB,dstRgn: RgnHandle);   A8E4;
P UnionRgn  (srcRgnA,srcRgnB,dstRgn: RgnHandle);  A8E5;
P DiffRgn   (srcRgnA,srcRgnB,dstRgn: RgnHandle);     A8E6;
P XorRgn    (srcRgnA,srcRgnB,dstRgn: RgnHandle);     A8E7;
F  EqualRgn  (rgnA,rgnB: RgnHandle): boolean;        A8E3;
F  EmptyRgn  (rgn: RgnHandle): boolean;              A8E2;
F  PtInRgn   (pt: Point; rgn: RgnHandle): boolean;   A8E8;
F  RectInRgn (r: Rect; rgn: RgnHandle): boolean;     A8E9;
```

Operations Graphiques sur des Regions

```
P FrameRgn  (rgn: RgnHandle);                        A8D2;
P PaintRgn  (rgn: RgnHandle);                        A8D3;
P EraseRgn  (rgn: RgnHandle);                        A8D4;
P InvertRgn (rgn: RgnHandle);                        A8D5;
P FillRgn   (rgn: RgnHandle; pat: Pattern);          A8D6;
```

Operations Graphiques sur des BitMaps

```
P  ScrollRect(dstRect:  Rect;  dh,dv:  integer;  updateRgn:
rgnHandle);                                  A8EF;
P CopyBits  (srcBits,dstBits: BitMap;
             srcRect,dstRect: Rect;
              mode: integer; maskRgn: RgnHandle);     A8EC;
```

Routines de PICTures

```
F  OpenPicture(picFrame: Rect): PicHandle;           A8F3;
P ClosePicture;                                      A8F4;
P DrawPicture(myPicture: PicHandle; dstRect: Rect);  A8F6;
P  PicComment(kind,dataSize:  integer;  dataHandle:  Handle);
A8F2;
P KillPicture(myPicture: PicHandle);                 A8F5;
```

Interface

```
P SetStdProcs(var procs: QDProcs); A8EA;
P StdText    (count: integer; textAddr: Ptr; numer,denom: Point);
                                          A882;
P StdLine    (newPt: Point);              A890;
P StdRect    (verb: GrafVerb; r: Rect); A8A0;
P StdRRect   (verb: GrafVerb; r: Rect; ovWd,ovHt: integer); A8AF;
```

```
P StdOval   (verb: GrafVerb; r: Rect);  A8B6;
P StdArc    (verb: GrafVerb; r: Rect; startAngle,arcAngle: integer);
                                        A8BD;
P StdPoly   (verb: GrafVerb; poly: PolyHandle);      A8C5;
P StdRgn    (verb: GrafVerb; rgn: RgnHandle);        A8D1;
P StdBits   (var srcBits: BitMap; var srcRect,dstRect: Rect;
              mode: integer; maskRgn: RgnHandle);  A8EB;
P StdComment (kind,dataSize: integer; dataHandle: Handle);
A8F1;
F StdTxMeas (count: integer; textAddr: Ptr;
             var numer,denom: Point; var info: FontInfo): integer;
A8ED;
P StdGetPic (dataPtr: Ptr; byteCount: integer); A8EE;
P StdPutPic (dataPtr: Ptr; byteCount: integer); A8F0;
```

Utilitaires QuickDraw

```
F GetPixel (h,v: integer): boolean; A865;
F Random:  integer;                 A861;
P StuffHex (thingptr: Ptr; s:Str255); A866;
P ForeColor (color: longint);         A862;
P BackColor (color: longint);         A863;
P ColorBit (whichBit: integer);       A864;
```

APPENDICE D

[Event Mgr, Os Utilities, VR Mgr, OS disp., core, queue]

OS Event Manager

F PostEvent(eventNum: integer; eventMsg: longint): OsErr;
P FlushEvents(whichMask,stopMask: integer);
P SetEventMask(theMask: integer);
F OSEventAvail(mask: integer; var theEvent: EventRecord): boolean;
F GetOSEvent(mask: integer; var theEvent: EventRecord): boolean;

ToolBox Event Manager

F EventAvail(mask:integer; var theEvent: EventRecord): boolean; A971;
F GetNextEvent(mask:integer; var theEvent: EventRecord): boolean; A970;
F StillDown:boolean; A973;
F WaitMouseUp:boolean; A977;
P GetMouse (var pt: Point); A972;
F TickCount:longint; A975;
F Button: boolean; A974;
P GetKeys(var k: keyMap); A976;
F GetDblTime: longint;
F GetCaretTime: longint;

Utilitaires de l'OS

F HandToHand(var theHndl: Handle): OsErr;
F PtrToXHand(srcPtr: Ptr; dstHndl: Handle; size: longint): OsErr;
F PtrToHand(srcPtr: Ptr; var dstHndl: Handle; size: longint): OsErr;
F HandAndHand(hand1,hand2: Handle): OsErr;
F PtrAndHand(ptr1: Ptr; hand2: Handle; size: longint): OsErr;
P SysBeep(duration: integer); A9C8;
P Environs(var rom,machine: integer);
P Restart;

vertical retrace mgr

F VInstall(VBLTaskPtr: QElemPtr): OsErr;
F VRemove(VBLTaskPtr: QElemPtr): OsErr;

Operating System Dispatcher

```
F  GetTrapAddress(trapNum: integer): longint;
P SetTrapAddress(trapAddr: longint; trapNum: integer);
```

utility core

```
F GetSysPPtr: SysPPtr;
F WriteParam: OsErr;
F  SetDateTime(time:  longint):OsErr;
F  ReadDateTime(var time: longint):OsErr;
P GetDateTime(var secs: longint);
P SetTime(d: DateTimeRec);
P GetTime(var d: DateTimeRec);
P Date2Secs(d: DateTimeRec; var s: longint);
P Secs2Date(s: longint; var d: DateTimeRec);
P Delay(numTicks: longint; var finalTicks: longint);
F  EqualString(str1,str2: Str255;
                    caseSens,diacSens: boolean):boolean;
P UprString(var theString: Str255; diacSens: boolean);
F  InitUtil: OsErr;
P UnLoadSeg(routineAddr: Ptr);  A9F1;
P ExitToShell;                  A9F4;
P GetAppParms(var apName: str255; var apRefNum: integer;
                    var apParam: Handle); A9F5;
P CountAppFiles(var message: integer; var count: integer);
P GetAppFiles(index: integer; var theFile: AppFile);
P ClrAppFiles(index: integer);
```

queue routines

```
P FInitQueue;
P Enqueue(qElement: QElemPtr; qHeader: QHdrPtr);
F  Dequeue(qElement: QElemPtr; qHeader: QHdrPtr): OsErr;
F  GetFSQHdr: QHdrPtr;
F  GetDrvQHdr: QHdrPtr;
F  GetVCBQHdr: QHdrPtr;
F  GetVBLQHdr: QHdrPtr;
F  GetEvQHdr: QHdrPtr;
F GetDCtlEntry(refNum: integer): DCtlHandle;
```

APPENDICE E [I/O, Sound, sys err, Util.]

```
F PBOpen(paramBlock: ParmBlkPtr; aSync: boolean): OsErr;
F PBClose(paramBlock: ParmBlkPtr; aSync: boolean): OsErr;
F PBRead(paramBlock: ParmBlkPtr; aSync: boolean): OsErr;
F PBWrite(paramBlock: ParmBlkPtr; aSync: boolean): OsErr;
F PBControl(paramBlock: ParmBlkPtr; aSync: boolean): OsErr;
F PBStatus(paramBlock: ParmBlkPtr; aSync: boolean): OsErr;
F PBKillIO(paramBlock: ParmBlkPtr; aSync: boolean): OsErr;
F PBGetVInfo(paramBlock: ParmBlkPtr; aSync: boolean): OsErr;
F PBGetVol(paramBlock: ParmBlkPtr; aSync: boolean): OsErr;
F PBSetVol(paramBlock: ParmBlkPtr; aSync: boolean): OsErr;
F PBFlushVol(paramBlock: ParmBlkPtr; aSync: boolean): OsErr;
F PBCreate(paramBlock: ParmBlkPtr; aSync: boolean): OsErr;
F PBDelete(paramBlock: ParmBlkPtr; aSync: boolean): OsErr;
F PBOpenRF(paramBlock: ParmBlkPtr; aSync: boolean): OsErr;
F PBRename(paramBlock: ParmBlkPtr; aSync: boolean): OsErr;
F PBGetFInfo(paramBlock: ParmBlkPtr; aSync: boolean): OsErr;
F PBSetFInfo(paramBlock: ParmBlkPtr; aSync: boolean): OsErr;
F PBSetFLock(paramBlock: ParmBlkPtr; aSync: boolean): OsErr;
F PBRstFLock(paramBlock: ParmBlkPtr; aSync: boolean): OsErr;
F PBSetFVers(paramBlock: ParmBlkPtr; aSync: boolean): OsErr;
F PBAllocate(paramBlock: ParmBlkPtr; aSync: boolean): OsErr;
F PBGetEOF(paramBlock: ParmBlkPtr; aSync: boolean): OsErr;
F PBSetEOF(paramBlock: ParmBlkPtr; aSync: boolean): OsErr;
F PBGetFPos(paramBlock: ParmBlkPtr; aSync: boolean): OsErr;
F PBSetFPos(paramBlock: ParmBlkPtr; aSync: boolean): OsErr;
F PBFlushFile(paramBlock: ParmBlkPtr; aSync: boolean): OsErr;
F PBMountVol(paramBlock: ParmBlkPtr): OsErr;
F PBUnMountVol(paramBlock: ParmBlkPtr): OsErr;
F PBEject(paramBlock: ParmBlkPtr): OsErr;
F PBOffLine(paramBlock: ParmBlkPtr): OsErr;
P AddDrive(drvrRefNum: integer; drvNum: integer; QEl:
drvQElPtr);
F FSOpen(fileName: Str255; vRefNum: integer;
          var refNum: integer): OsErr;
F FSClose(refNum: integer): OsErr;
F FSRead(refNum: integer; var count: longint; buffPtr: Ptr): OsErr;
F FSWrite(refNum: integer; var count: longint; buffPtr: Ptr): OsErr;
F Control(refNum: integer; csCode: integer;
          csParamPtr: Ptr): OsErr;
F Status(refNum: integer; csCode: integer;
          csParamPtr: Ptr): OsErr;
F KillIO(refNum: integer): OsErr;
```

appels concernant les volumes

```
F  GetVInfo (drvNum: integer; volName: StringPtr;
                var vRefNum: integer;
                var FreeBytes: longint): OsErr;
F  GetFInfo(fileName: Str255; vRefNum: integer;
                var FndrInfo: FInfo):OsErr;
F  GetVol(volName: StringPtr; var vRefNum: integer):OsErr;
F  SetVol(volName: StringPtr; vRefNum: integer): OsErr;
F  UnMountVol(volName: StringPtr; vRefNum: integer):OsErr;
F  Eject(volName: StringPtr; vRefNum: integer): OsErr;
F  FlushVol(volName: StringPtr; vRefNum: integer):OsErr;
```

pour fichiers non ouverts

```
F  Create(fileName: Str255; vRefNum: integer; creator: OSType;
            fileType: OSType):OsErr;
F  FSDelete(fileName: Str255; vRefNum: integer):OsErr;
F  OpenRF(fileName: Str255; vRefNum: integer;
            var refNum: integer): OsErr;
F  Rename(oldName: Str255; vRefNum: integer;
            newName: Str255):OsErr;
F  SetFInfo(fileName: Str255; vRefNum: integer;
            FndrInfo: FInfo):OsErr;
F  SetFLock(fileName: Str255; vRefNum: integer):OsErr;
F  RstFLock(fileName: Str255; vRefNum: integer):OsErr;
```

pour fichiers ouverts

```
F  Allocate(refNum: integer; var count: longint):OsErr;
F  GetEOF(refNum: integer; var LogEOF: longint):OsErr;
F  SetEOF(refNum: integer; LogEOF: longint):OsErr;
F  GetFPos(refNum: integer; var filePos: longint):OsErr;
F   SetFPos(refNum: integer; posMode: integer; posOff:
longint):OsErr;
F  GetVRefNum(fileRefNum: integer):integer;
```

Ser. Driver Interface

```
F  OpenDriver(name: Str255; var drvrRefNum: integer): OsErr;
F  CloseDriver(refNum:  integer):OsErr;
F  SerReset(refNum: integer; serConfig: integer): OSErr;
F   SerSetBuf(refNum: integer; serBPtr: Ptr; serBLen: integer):
OSErr;
F  SerHShake(refNum: integer; flags: SerShk): OSErr;
F  SerSetBrk(refNum: integer): OSErr;
F  SerClrBrk(refNum: integer): OSErr;
F  SerGetBuf(refNum: integer; var count: longint): OSErr;
```

```
F  SerStatus(refNum: integer; var serSta: SerStaRec): OSErr;
F DiskEject(drvnum: integer): OSErr;
F SetTagBuffer(buffPtr: Ptr): OSErr;
F DriveStatus(drvNum: integer; var status: DrvSts): OSErr;
F  RamSDOpen(whichPort: SPortSel): OSErr;
P  RamSDClose(whichPort: SPortSel);
```

Sound Driver

```
P SetSoundVol(level: integer);
P GetSoundVol(var level: integer);
P StartSound(synthRec: Ptr; numBytes: longint;
                CompletionRtn: ProcPtr);
P StopSound;
F SoundDone: boolean;
```

system error handler

```
P SysError(errorCode: integer);
```

Utilitaires généraux

```
F  BitAnd   (long1,long2: longint): longint; A858;
F  BitOr (long1,long2: longint): longint;    A85B;
F  BitXor   (long1,long2: longint): longint;  A859;
F  BitNot   (long: longint): longint;          A85A;
F  BitShift (long: longint; count: integer): longint;   A85C;
F  BitTst   (bytePtr: Ptr; bitNum: longint): boolean;   A85D;
P BitSet   (bytePtr: Ptr; bitNum: longint); A85E;
P BitClr   (bytePtr: Ptr; bitNum: longint); A85F;
P LongMul  (a,b: longint; var dst: Int64Bit);  A867;
F  FixMul   (a,b: Fixed): Fixed; A868;
F  FixRatio (numer,denom: integer): Fixed;  A869;
F  HiWord   (x: longint): integer;  A86A;
F  LoWord   (x: longint): integer;  A86B;
F  FixRound (x: Fixed): integer;    A86C;
P PackBits   (var srcPtr,dstPtr: Ptr; srcBytes: integer); A8CF;
P UnPackBits (var srcPtr,dstPtr: Ptr; dstBytes: integer); A8D0;
F SlopeFromAngle(angle: integer): Fixed;  A8BC;
F AngleFromSlope(slope: Fixed): integer;  A8C4;
F DeltaPoint(ptA,ptB: Point): longint;  A94F;
F  NewString(theString:Str255): StringHandle;  A906;
P SetString(theString:StringHandle; strNew: Str255);  A907;
F  GetString(stringID:  integer): StringHandle;A9BA;
P GetIndString(var theString: str255; strListID: integer;
                index: integer);
F Munger(h: Handle; offset: longint; ptr1: Ptr; len1: longint;
          ptr2: Ptr; len2: longint): longint;  A9E0;
```

```
F  GetIcon(iconID:    integer):  Handle; A9BB;
P PlotIcon(theRect: Rect; theIcon: Handle);  A94B;
F  GetCursor (cursorID: integer):  CursHandle; A9B9;
F  GetPicture(picID: integer):  PicHandle;  A9BC;
P GetIndPattern(var thePat: Pattern; patListID: integer;
                    index: integer);
F  GetPattern(patID: integer):  PatHandle; A9B8;
P ShieldCursor(shieldRect: Rect; offsetPt: Point); A855;
```

APPENDICE F

[Font Mgr, Window Mgr, TextEdit]

Font Manager

```
P InitFonts;                                    A8FE;
P GetFontName(familyID: integer; var theName: Str255);  A8FF;
P GetFNum(theName: Str255; var familyID: integer); A900;
P SetFontLock(lockFlag: boolean); A903;
F  FMSwapFont(inRec: FMInput): FMOutPtr; A901;
F  RealFont(famID: integer; size: integer): boolean; A902;
```

Window Manager

```
P ClipAbove(window: WindowPeek); A90B;
P  PaintOne(window:  WindowPeek;  clobbered:  RgnHandle);
A90C;
P   PaintBehind(startWindow:   WindowPeek;   clobbered:
RgnHandle);                                     A90D;
P SaveOld(window: WindowPeek); A90E;
P DrawNew(window: WindowPeek; fUpdate: boolean);   A90F;
P CalcVis(window: WindowPeek); A909;
P CalcVisBehind(startWindow: WindowPeek;
                clobbered: RgnHandle); A90A;
P ShowHide(window: WindowPtr; showFlag: boolean);  A908;
F  CheckUpdate(var theEvent: EventRecord): boolean; A911;
P GetWMgrPort(var wPort: GrafPtr); A910;
P InitWindows;                                  A912;
F  NewWindow(wStorage:   Ptr;
             boundsRect: Rect;
             title:     Str255;
             visible:   boolean;
             theProc:   integer;
             behind:    WindowPtr;
             goAwayFlag: boolean;
             refCon:    longint): WindowPtr; A913;
P DisposeWindow (theWindow: WindowPtr); A914;
P CloseWindow   (theWindow: WindowPtr); A92D;
P MoveWindow    (theWindow: WindowPtr; h,v: integer;
                   BringToFront: boolean); A91B;
P SizeWindow    (theWindow: WindowPtr; width,height: integer;
                  fUpdate: boolean); A91D;
F  GrowWindow    (theWindow: windowPtr;
                   startPt: Point; bBox: Rect): longint;  A92B;
P DragWindow    (theWindow: WindowPtr;
                  startPt: Point; boundsRect: Rect); A925;
```

```
P ShowWindow    (theWindow: WindowPtr); A915;
P HideWindow    (theWindow: WindowPtr);  A916;
P SetWTitle     (theWindow: WindowPtr; title: Str255); A91A;
P GetWTitle     (theWindow: WindowPtr; var title: Str255); A919;
P HiliteWindow  (theWindow: WindowPtr; fHiLite: boolean); A91C;
P BeginUpdate   (theWindow: WindowPtr); A922;
P EndUpdate     (theWindow: WindowPtr); A923;
P SetWRefCon    (theWindow: WindowPtr; data: longint);  A918;
F  GetWRefCon   (theWindow: WindowPtr): longint;  A917;
P  SetWindowPic  (theWindow: WindowPtr; thePic: PicHandle);
A92E;
F  GetWindowPic  (theWindow: WindowPtr): PicHandle; A92F;
P BringToFront  (theWindow: WindowPtr); A920;
P SendBehind    (theWindow,behind: WindowPtr);  A921;
F  FrontWindow:                          WindowPtr; A924;
P SelectWindow  (theWindow: WindowPtr); A91F;
F  TrackGoAway   (theWindow: WindowPtr; thePt: Point): boolean;
                                         A91E;
P DrawGrowIcon  (theWindow: WindowPtr); A904;

P ValidRect     (goodRect: Rect);  A92A;
P ValidRgn      (goodRgn: RgnHandle);  A929;
P InvalRect     (badRect: Rect); A928;
P InvalRgn      (badRgn:  RgnHandle); A927;
F  FindWindow    (thePoint: Point;
                    var  theWindow: WindowPtr): integer;  A92C;
F  GetNewWindow  (windowID: integer; wStorage: Ptr;
                      behind: WindowPtr): WindowPtr;    A9BD;
F  PinRect(theRect: Rect; thePt: Point): longint;   A94E;
F  DragGrayRgn(theRgn: RgnHandle; startPt: Point; boundsRect,
              slopRect: Rect;axis: integer;
              actionProc: ProcPtr): longint; A905;
```

TextEdit

```
P TEActivate( h: TEHandle );  A9D8;
P TECalText( h: TEHandle );  A9D0;
P TEClick( pt: Point; extend: boolean; h: TEHandle );  A9D4;
P TECopy( h: TEHandle );      A9D5;
P TECut( h: TEHandle );       A9D6;
P TEDeActivate( h: TEHandle ); A9D9;
P TEDelete( h: TEHandle );    A9D7;
P TEDispose( h: TEHandle ); A9CD;
P TEIdle( h: TEHandle );      A9DA;
P TEInit;                      A9CC;
P TEKey( key: char; h: TEHandle ); A9DC;
F  TENew( dest, view: Rect ): TEHandle; A9D2;
P TEPaste( h: TEHandle );     A9DB;
```

```
P TEScroll( dh, dv: integer; h: TEHandle );   A9DD;
P TESetSelect( selStart, selEnd: longint; h: TEHandle );     A9D1;
P TESetText( inText: Ptr; textLength: longint; h: TEHandle );
A9CF;
P TEInsert( inText: Ptr; textLength: longint; h: TEHandle );
A9DE;
P TEUpdate( rUpdate: Rect; h: TEHandle );  A9D3;
P TESetJust( just: integer; h: TEHandle );     A9DF;
F  TEGetText( h: TEHandle ): CharsHandle; A9CB;
F TEScrapHandle: Handle;
F TEGetScrapLen: longint;
P TESetScrapLen(length: longint);
F TEFromScrap: OsErr;
F TEToScrap: OsErr;

{ appelez vos propres routines PasWordBreak }
{ PasClikLoop }

{AsmWordBreak - fixe le stack
        F PasWordBreak(text: Ptr; index: integer):boolean; }

P AsmWordBreak;

{ AsmClikLoop -  fixe le stack avant un appel à PasClikLoop }

P AsmClikLoop;
P TextBox( inText: Ptr; textLength: longint; r: Rect; style: integer );
                                A9CE;
```

APPENDICE G

[Resource Mgr, Control Mgr]

Resource Manager

```
F  InitResources: integer;                A995;
P RsrcZoneInit;                           A996;
P CreateResFile(fileName:  Str255); A9B1;
F  OpenResFile(fileName: Str255): integer; A997;
P UseResFile(refNum:  integer); A998;
F  GetResFileAttrs(refNum:  integer): integer; A9F6;
P SetResFileAttrs(refNum:  integer;  attrs:  integer);  A9F7;
P UpdateResFile(refNum: integer); A999;
P CloseResFile(refNum:  integer); A99A;
P SetResPurge(install:  boolean); A993;
P SetResLoad(AutoLoad:  boolean); A99B;
F  CountResources(theType:  ResType): integer; A99C;
F  GetIndResource(theType: ResType; index:  integer): Handle;
                                  A99D;
F  CountTypes:  integer;  A99E;
P GetIndType(var theType:  ResType; index:  integer);   A99F;
F  UniqueID(theType:  ResType):  integer;  A9C1;
F  GetResource(theType: ResType; ID: integer): Handle;  A9A0;
F   GetNamedResource(theType:   ResType;  name:  Str255):
Handle;
                                  A9A1;
P LoadResource(theResource: Handle);  A9A2;
P ReleaseResource(theResource:  Handle);  A9A3;
P DetachResource(theResource:  Handle); A992;
P ChangedResource(theResource:  Handle); A9AA;
P WriteResource(theResource:  Handle); A9B0;
F  HomeResFile(theResource: Handle): integer; A9A4;
F  CurResFile:  integer; A994;
F  GetResAttrs(theResource: Handle): integer; A9A6;
P SetResAttrs(theResource: Handle; attrs: integer); A9A7;
P GetResInfo(theResource:  Handle;
              var theID: integer;
              var theType:  ResType;
              var name: Str255); A9A8;
P SetResInfo(theResource:  Handle;
              theID: integer;
              name: Str255); A9A9;
P AddResource(theResource:  Handle;
               theType: ResType;
               theID: integer;
               name: Str255);  A9AB;
P AddReference(theResource:  Handle;
```

```
                theID: integer; name: Str255); A9AC;
P RmveResource(theResource:  Handle); A9AD;
P RmveReference(theResource:  Handle);  A9AE;
F SizeResource(theResource: Handle): longint; A9A5;
F  ResError:  integer; A9AF;
```

Control Manager

```
F  NewControl(curWindow:          windowPtr;
              boundsRect: Rect;
              title:     Str255;
              visible:    boolean;
              value:      integer;
              min:       integer;
              max:        integer;
              contrlProc: integer;
              refCon:     longint): ControlHandle;  A954;
P DisposeControl (theControl: ControlHandle); A955;
P KillControls                          (theWindow:     WindowPtr);
A956;
P MoveControl (theControl: ControlHandle; h,v: integer); A959;
P SizeControl (theControl: ControlHandle; w,h: integer); A95C;
P DragControl (theControl: ControlHandle; startPt: Point;
                  bounds: Rect; slopRect: Rect; axis:integer); A967;
P ShowControl (theControl: ControlHandle); A957;
P HideControl (theControl: ControlHandle); A958;
P SetCTitle (theControl: ControlHandle; title: Str255); A95F;
P GetCTitle (theControl: ControlHandle; var title: Str255); A95E;
P HiliteControl (theControl: ControlHandle; hiliteState: integer);
                                A95D;
P SetCRefCon (theControl: ControlHandle; data: longint); A95B;
F  GetCRefCon (theControl: ControlHandle): longint;  A95A;
P  SetCtlValue  (theControl: ControlHandle; theValue: integer);
A963;
F  GetCtlValue (theControl: ControlHandle): integer;  A960;
F  GetCtlMin (theControl: ControlHandle): integer;  A961;
F  GetCtlMax (theControl: ControlHandle): integer;  A962;
P  SetCtlMin  (theControl: ControlHandle; theValue: integer);
A964;
P  SetCtlMax  (theControl: ControlHandle; theValue: integer);
A965;
F  GetCtlAction (theControl: ControlHandle): ProcPtr;  A96A;
P  SetCtlAction  (theControl: ControlHandle; newProc: ProcPtr);
A96B;
F  TestControl (theControl: ControlHandle; thePt: Point): integer;
                                A966;
F  TrackControl (theControl:ControlHandle; thePt: Point;
                  actionProc:ProcPtr):integer;  A968;
```

```
F  FindControl (thePoint: Point;
                theWindow: WindowPtr;
                var theControl: ControlHandle): integer;  A96C;
P DrawControls (theWindow: WindowPtr); A969;
F  GetNewControl (controlID: integer; owner: WindowPtr):
                                                ControlHandle;  A9BE;
```

APPENDICE H

[Dialog Mgr, Desk Mgr, Menu Mgr, scrap Mgr]

Dialog Manager

```
P InitDialogs(resumeProc: ProcPtr); A97B;
F GetNewDialog(dialogID: integer;
               wStorage: Ptr;
               behind: WindowPtr): DialogPtr; A97C;
F NewDialog(wStorage: Ptr;
            boundsRect: Rect;
            title: Str255;
            visible: boolean;
            theProc: integer;
            behind: WindowPtr;
            goAwayFlag: boolean;
            refCon: longint;
            itmLstHndl: Handle): DialogPtr; A97D;
F IsDialogEvent(event: EventRecord): boolean; A97F;
F DialogSelect( event: EventRecord;
                var theDialog: DialogPtr;
                var itemHit: integer): boolean; A980;
P ModalDialog( filterProc: ProcPtr; var itemHit: integer); A991;
P DrawDialog(dialog: DialogPtr); A981;
P CloseDialog(dialog: DialogPtr); A982;
P DisposDialog(dialog: DialogPtr); A983;
F Alert(alertID: integer;
        filterProc: ProcPtr): integer; A985;
F StopAlert(alertID: integer;
            filterProc: ProcPtr): integer; A986;
F NoteAlert(alertID: integer;
            filterProc: ProcPtr): integer; A987;
F CautionAlert(alertID: integer;
               filterProc: ProcPtr): integer; A988;
P CouldAlert(alertID: integer); A989;
P FreeAlert(alertID: integer); A98A;
P CouldDialog(DlgID: integer); A979;
P FreeDialog(DlgID: integer); A97A;
P ParamText(cite0, cite1, cite2, cite3:Str255); A98B;
P ErrorSound(sound: ProcPtr); A98C;
P GetDItem(dialog: DialogPtr;
           itemNo: integer;
           var kind: integer;
           var item: Handle;
           var box: Rect); A98D;
P SetDItem(dialog: DialogPtr;
           itemNo: integer;
```

```
                kind: integer;
                item: Handle;
                box: Rect); A98E;
P SetIText(item:      Handle;
                text:     Str255); A98F;
P GetIText(item:      Handle;
                var text:  Str255); A990;
P SelIText(dialog:   DialogPtr;
                itemNo:  integer;
                startSel, endSel: integer ); A97E;
```

{ **utilisées en Pascal** }

```
F GetAlrtStage: integer;
P ResetAlrtStage;
P DlgCut(dialog: DialogPtr);
P DlgPaste(dialog: DialogPtr);
P DlgCopy(dialog: DialogPtr);
P DlgDelete(dialog: DialogPtr);
P SetDAFont(fontNum: integer);
```

Desk Manager

```
F  SystemEvent(myEvent: EventRecord): boolean; A9B2;
P SystemClick(theEvent: EventRecord; theWindow: windowPtr);
                                        A9B3;
P SystemTask;                           A9B4;
P SystemMenu(menuResult: longint); A9B5;
F  SystemEdit(editCode: integer): boolean; A9C2;
F  OpenDeskAcc(theAcc: Str255): integer; A9B6;
P CloseDeskAcc(refNum: integer); A9B7;
```

Menu Manager

```
P InitMenus; A930;
F  NewMenu(menuID: integer; menuTitle: Str255): menuHandle;
                                        A931;
F  GetMenu(rsrcID:  integer): MenuHandle; A9BF;
P DisposeMenu(menu: menuHandle); A932;
P AppendMenu(menu: menuHandle; data: str255);  A933;
P InsertMenu (menu: MenuHandle; beforeId: integer); A935;
P DeleteMenu (menuId: integer);  A936;
P DrawMenuBar;                          A937;
P ClearMenuBar;                         A934;
F  GetMenuBar:Handle;                   A93B;
F  GetNewMBar(menuBarID: integer): Handle;  A9C0;
P SetMenuBar(menuBar: Handle); A93C;
F  MenuSelect (startPt: Point): longint;  A93D;
```

```
F MenuKey   (ch: char): longint; A93E;
P HiLiteMenu (menuId: integer); A938;
P SetItem     (menu: menuHandle; item: integer; itemString:
Str255);                                  A947;
P GetItem   (menu: menuHandle; item: integer;
              var itemString: Str255); A946;
P EnableItem (menu: menuHandle; item: integer); A939;
P DisableItem(menu: menuHandle; item: integer); A93A;
P CheckItem   (menu: menuHandle; item: integer; checked:
boolean);                                 A945;
P SetItemIcon(menu: menuHandle; item: integer; iconNum: Byte);
                                  A940;
P GetItemIcon(menu: menuHandle; item: integer; var iconNum:
Byte);                                    A93F;
P SetItemStyle(menu: menuHandle; item: integer; styleVal:
Style);                                   A942;
P GetItemStyle(menu: menuHandle; item: integer;
                  var styleVal: Style);  A941;
P SetItemMark(menu: menuHandle; item: integer; markChar:
char);                                    A944;
P GetItemMark(menu: menuHandle; item: integer;
                  var markChar: char);  A943;
P SetMenuFlash(flashCount: integer);  A94A;
P FlashMenuBar(menuID: integer); A94C;
F  GetMHandle(menuID: integer): menuHandle; A949;
F  CountMItems(menu:  menuHandle): integer;  A950;
P AddResMenu(menu: menuHandle; theType:ResType);   A94D;
P InsertResMenu(menu: menuHandle;
                    theType:ResType; afterItem: integer);     A951;
P CalcMenuSize(menu:menuHandle);  A948;
```

Scrap Manager

```
F  GetScrap( hDest: Handle; what: ResType;
              var offset: longint ): longint;  A9FD;
F  InfoScrap: pScrapStuff;          A9F9;
F  LoadScrap: longint;              A9FB;
F  PutScrap( length: longint; what: ResType;
               source: Ptr): longint;  A9FE;
F  UnloadScrap: longint;            A9FA;
F  ZeroScrap: longint;              A9FC;
```

package manager

```
P InitAllPacks;                     A9E6;
P InitPack(packID: integer);        A9E5;
```

Imprimerie de la Manutention à Mayenne
Dépôt légal : septembre 1986
N° d'Editeur : 4554

PROGRAMMES SOURCE

EXPLIQUES LIGNE PAR LIGNE

EXEMPLES ECRITS EN:

PASCAL, "C", FORTH, BASIC

LA TOOLBOX QUICKDRAW ORD4

HANDLES ^^ POINTERS TICKS

LES MANAGERS LES EVENTS

YANN FAIN est avant tout un mathématicien. Cependant, il a commencé à programmer en 1970, et dirige à présent une société d'informatique. Dans ce livre, il vous montre comment **mieux vous servir** de votre Macintosh™, et comment le **programmer** en Pascal, en "C", en Forth et en Basic. De nombreux exemples commentés sont fournis. Vous apprendrez, dans ces lignes, à personnaliser vos programmes, et à faire passer vos données d'une application à une autre sans passer par les fastidieux Couper-Coller, à réparer les Finder défectueux, et à programmer le Macintosh™.

Yann FAIN a aussi animé de nombreux **cours** de développement sur diverses machines, dont le Macintosh™. Rares sont ceux qui savent expliquer.

Yann FAIN est de ceux-là.

Couverture JÉRÔME LO MONACO

EYROLLES

8341

EYROLLES

USAGES SPECIAUX & PROGRAMMATION DU MACINTOSH EN PASCAL ET C

YANN FAIN